Load-n-Go: Fast Approximate Join Visualizations That Improve Over Time

Marianne Procopio, Carlos Scheidegger, Eugene Wu, Remco Chang

Abstract—Visual exploratory analysis of large-scale databases often relies on precomputed query results in order to guarantee interactivity with the visualization system. This is especially true when the query requires joining tables across the database since join is one of the most computationally expensive operations and in the worst case requires joining every row of one table to every row of the other table. Advances in approximate query processing enable quick look summary statistics, but with limitations to the types and complexities of the queries. A recent advancement in approximate query processing is a technique called Wander Join, that performs random walks across these joins, resulting in faster convergence on aggregation values. However, this online aggregation technique is not tailored for visualization tasks that often involve filtering on one or more conditions or viewing aggregation values across multiple groups such as in bar charts. In both cases, the convergence rate is slowed since more samples are needed in order to find records that pass the filters, resulting in the user waiting longer for a confident result. To address this issue, we propose a generalization of the Wander Join algorithm that improves the convergence rate for visualization queries involving filtering and viewing aggregation. We implemented this improved version of Wander Join that we call Load-n-Go and compared it to the original, specifically in the context of visual analysis tasks. Our evaluation finds that our algorithm outperforms Wander Join by reducing the sample complexity. Load-n-Go requires up to 50% fewer samples for group by queries, and up to 85% fewer samples for filtering queries. Such reduced sampling complexity can represent up to 2x and 6x speedups respectively for visual exploratory systems using the Load-n-Go algorithm.

1 Introduction

Responsive visual exploration typically requires that the underlying data be available at interactive speeds. This requirement can be easily met when data resides in memory, but such a solution is not scalable. As the amount of data increases, the data must be stored on disk or in a remote database. As a result, the queries used to populate the visualization can take minutes, hours or longer to return, resulting in long wait times between each of the user's interactions and diminishing the user's ability to quickly explore the data.

This lag is exacerbated if the analysis queries involve joining data across multiple database tables. For example, consider the following SQL query that finds the average customer spending per region.

```
SELECT   AVG(orders.order_total), location.region
FROM     orders, customers, location
WHERE    orders.customerID = customer.customerID AND
         customer.locationID = location.locationID
GROUP BY location.region
```

The query must read data from three separate tables, "orders", "customers" and "locations", combine them together using the expressions in the `WHERE` clause, partition the resulting data by `location.region`, and finally compute the average `order_total` for each region. `JOIN` operations are well known to be costly [35]. To compute fully accurate results, the join operation requires at minimum full scans of both input tables, and naive implementations take quadratic time in the sizes of the two tables.

1.1 Online Aggregation, Ripple Join, and Wander Join

Although the classic approach to improve query response times is to precompute all possible results, this can require a long wait time before the user sees any result. In contrast, it is often desirable to load new datasets and immediately start performing analyses. To this end, there has been recent interest in *online aggregation* algorithms that do not require precomputation and instead select samples "online" when the analyst submits a new query [19, 25, 3]. Such algorithms can provide real-time (approximate) query results along with their associated confidence intervals as the systems draw samples from the database while executing the query. The goal of this work is to reduce the number of samples necessary to achieve a desired error bound at a given confidence level. Reducing this *sample complexity* potentially means that the number of input records that need to be read can be reduced by orders of magnitude as compared to fully reading the input tables.

Extending the original online aggregation algorithm [19], Ripple Join was designed specifically to support `JOIN` queries [17]. Unfortunately, Ripple Join requires sample sizes that can be impractical for the interactive needs of visualization applications. For a query that joins tables `A` and `B`, the algorithm samples records from each table independently, and then checks whether or not they satisfy the `JOIN` condition, but often the join of two randomly chosen records is highly unlikely to pass. Thus, the number of records that need to be sampled can be very large before the CI converges to a satisfactory level and the approximate query result is sufficiently close to the ground truth.

The Wander Join algorithm addresses this issue [26]. Instead of sampling randomly from each table like Ripple Join, Wander Join samples randomly from table `A` and then chooses a record from table `B` that the record can join with. Wander Join does this by modeling the `JOIN`s as a *join graph* and walking along edges to find a valid record to join against in the next table. This method increases the convergence rate of the query, resulting in less wait time for the user to achieve a higher confidence estimate.

Although Wander Join is a significant improvement over prior techniques for executing `JOIN` queries, it is not directly suited for interactive visualization. A key limitation is that it draws samples independently of the `WHERE` and `GROUP BY` filters in the query. This problem is exacerbated when few records are able to satisfy the filters, as is the case when the user is interested in a small segment of the dataset, or when the groups in the `GROUP BY` query exhibit skew. Crucially, these two conditions are common in interactive visualization scenarios.

We evaluated Wander Join in interactive visualization settings where users want to dynamically filter on expensive `JOIN` queries, and extended its techniques to reduce the sample complexity for this class of query workloads. This extension of Wander Join addresses the needs of interactive visualizations where users often select subsets of data resulting in executing expensive `JOIN` queries with highly selective filters.

- *Marianne Procopio and Remco Chang are with Tufts University. Email: Marianne.Procopio@tufts.edu, remco@cs.tufts.edu.*
- *Carlos Scheidegger is with the University of Arizona. Email: cscheid@email.arizona.edu.*
- *Eugene Wu is with Columbia University. Email: ewu@cs.columbia.edu.*

1.2 Load-n-Go

At its core, our proposed system, Load-n-Go, is a progressive visualization system that improves the accuracy of the visualization over time. It uses an extension to Wander Join that addresses the limitations described above. The key algorithmic insight in Load-n-Go is to take the query filters into account when drawing samples from the database by prioritizing samples that are more likely to satisfy the filters. To achieve this, we integrate the idea of importance sampling [18] into Wander Join and describe how to perform this prioritization.

In evaluating and comparing with Wander Join, our technique outperforms the original algorithm for queries involving `WHERE` and `GROUP BY` clauses. By skipping samples that do not pass the filter, we achieved a convergence rate that is up to 6 times faster than Wander Join, meaning Wander Join requires 6 times more samples than Load-n-Go to achieve the same confidence interval. When executing `GROUP BY` queries, we require up to 50% fewer samples than Wander Join when group membership is unevenly distributed.

2 Related Work

There is a clear need for highly responsive visual interfaces that can explore large databases. There has been recent interest in co-designing interactive visualization with the underlying data processing systems that compute the results rendered onscreen [41, 42].

The most relevant work has been in progressive visualization systems that quickly render results with wide confidence intervals (less accuracy) and improve the accuracy over time. Systems such as sampleAction [15] were instrumental in showing that users benefited from seeing immediate results alongside incrementally improving accuracies, and that users felt more empowered to actively explore their data instead of waiting for queries to complete. Their results highlight the importance of researching effective ways to visualize approximate and improving results [13, 43, 30, 33], as well as a need for further study.

Towards scalable systems, Im et al. [21], Pansare et al. [32] and others [8, 5, 22] developed systems for running online aggregation in a distributed manner. They take advantage of MapReduce and other distributed computation infrastructure, but are limited to queries on single tables, and do not provide user controls to adjust the resources that the system devotes to computing higher accuracy values for different groups in the visualization.

Beyond online aggregation, there are a number of techniques that can be used to support visual exploration and analysis of big data. In addition to sampling [1, 2, 9, 23, 27, 12, 11, 14, 13, 34, 37, 21] and precomputation [38, 28, 29, 31, 40], researchers have also explored the use of predictive prefetching [10, 7, 4, 6] and specialized databases such as column stores [39] and in-memory databases [24]. Additionally, some systems use a combination of these techniques [4]. For a more comprehensive review of these techniques and their use in visualization, refer to the survey by Godfrey et al. [16].

3 Wander Join Algorithm

Wander Join represents the state of the art in online aggregation across `JOIN`s. Like most other approximate query processing techniques, it provides online aggregation support for computing summary statistics such as sums and counts. More importantly, it improves on the convergence rate over Ripple Join by intelligently selecting which records to join with. It does this by modeling the `JOIN`s as a join graph and walking along edges to find valid joined data. Figure 1 shows an example join graph for joining tables A, B and C. In this graph, each row (e.g. a1, a2,...) represents a record in one of the tables, and an edge between two rows means that the two records can be joined in a natural join. Wander Join selects one of the records (a1-a5) in Table A, then randomly selects a path to Table B, and repeats for Table C.

An estimate of the aggregate value is returned after each walk. Since random walks will not return the uniform distribution needed to generate an unbiased estimate, Wander Join uses the *Horvitz-Thompson estimator* [20]. This removes bias by dividing the value to be aggregated by the probability of having chosen the path taken to reach that value. For example, assuming the tables A, B and C can be joined across their common dimensions:

$$A(d1,d2) \bowtie B(d2,d3) \bowtie C(d3,d4) \tag{1}$$

We run a query to sum over the dimension $d4$. Wander Join selects a2 from the join graph below. It then randomly selects one of the edges leading from a2 to Table B, resulting in either b1 or b2 as the next record. If it picks b2 then c1, c2 or c3 is chosen next according to the graph. The probability of selecting this path is $\frac{1}{5} \times \frac{1}{2} \times \frac{1}{3}$. Let $v(d4)$ be the value of the $d4$ attribute for this sample. The *Horvitz-Thompson estimator* would return: $v(d4)/(\frac{1}{5} \times \frac{1}{2} \times \frac{1}{3})$.

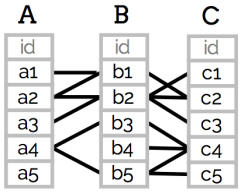

Fig. 1. In Wander Join, `JOIN`s are modeled as a graph, and random walks are taken along valid paths to select a sample.

A walk will fail if a record does not pass a filter specified in the query. In this case, the aggregate value is treated as 0 to keep the *Horvitz-Thompson estimator* unbiased, since this sample is in the probability space of the distribution. However, this slows convergence of the estimate. For the rest of this paper, we treat the convergence rate as measuring the reduction of the estimate's relative standard error:

$$\eta = \frac{z\frac{\sqrt{v}}{\sqrt{N}}}{E} \tag{2}$$

where v is the variance of the estimate, E, and N is the number of samples taken ($\frac{\sqrt{v}}{\sqrt{N}}$ is the standard error of the estimate). z is the z-score for the half width of the given confidence level. As the number of samples increases, the relative error approaches 0. The relative error allows us to compare convergence rates over different queries.

3.1 Using Wander Join In Visual Exploration

In order to evaluate Load-n-Go, we implemented the Wander Join algorithm and compared its performance to Load-n-Go. To ensure that the comparison is fair, we: (1) evaluated the performance of the two algorithms based on the number of samples needed to reach convergence instead of clock time where language, experimental platform, etc. can affect the results, (2) used the same experiments described in the original Wander Join paper, and (3) extended the evaluation to include queries relevant to interactive visualization where filtering using the `WHERE` and the `GROUP BY` clauses are common.

3.1.1 Data

Similar to the original Wander Join paper, we evaluated the performance of Wander Join using the TPC-H benchmark, a synthetically generated dataset that simulates a data warehouse, and includes a set of queries that represent common analysis queries by business analysts. For our evaluation, we used four sizes of TPC-H data: 1MB, 10MB, 100MB, and 2GB. However, we found that regardless of the data size, the performance profile remains the same, likely due to the nature that TPC-H data is generated by drawing from an even distribution. As a result, we only report the evaluation results using the 10MB dataset to reduce the impact of disk access delays during testing.

3.1.2 Validation Experiment

To verify that the accuracy of our Wander Join implementation is consistent with the original, we ran a validation experiment comparing performances for the following query to reach 95% confidence level:

```
SELECT sum(l_quantity)
FROM part, lineitem
WHERE part.p_partkey = lineitem.l_partkey
```

The results of this experiment show that both implementations of Wander Join required the same number of samples to achieve the same confidence interval, thus validating that our implementation is faithful to the original.

3.1.3 Evaluating Common Visualization Queries

We tested Wander Join's performance with filtering. First we tested the performance of Wander Join with queries involving `WHERE` clauses:

```
SELECT sum(l_quantity)
FROM part, lineitem
WHERE part.p_partkey = lineitem.l_partkey
AND part.p_size <= X
AND lineitem.l_quantity <= Y
```

This query returns the number of total parts sold for parts with sizes smaller than or equal to *X* and when the number sold per order is *Y* or less. *X* and *Y* were varied to filter out 0% to 99% of all rows in the full `JOIN`. If a filter is highly selective, most records do not pass the filter conditions (meaning the selective percentage is the percent of rows that are rejected by the filter conditions compared to the full `JOIN` without filtering). The higher the selective percentage, the fewer number of records that pass the filters.

Figure 3 shows the relative error for each level of filtering after 10,000 samples were taken for Wander Join. Note that when the selective filtering percentage is at 90% (the `WHERE` clause in the query filters out 90% of the data), the uniform sampling approach of Wander Join can be inefficient. This is because most of the samples drawn by Wander Join in this highly selective query will be considered a "failed walk" as these samples do not satisfy the `WHERE` condition.

For Wander Join to achieve 0.01 relative error, 208,000 samples would be needed. In contrast, the full `JOIN` of this query requires 60,175 rows. It is clear that in this case, the effectiveness of Wander Join is not only significantly reduced but Wander Join is over 3 times slower than running the full `JOIN`.

Our last set of testing was to evaluate Wander Join's performance for `GROUP BY` queries. We ran the same query as in the filter tests, but without the filter conditions and grouped on part size. We limited the dataset to have 40 different part sizes:

```
SELECT sum(l_quantity)
FROM part, lineitem
WHERE part.p_partkey = lineitem.l_partkey
GROUP BY part.p_size
```

This resulted in 40 groups, with an even distribution of the 60,175 records across all groups. It took approximately 25,000 total samples for each group to reach a maximum of .05 relative error. However, there are many situations where the records are not evenly distributed across all groups, such as charting sales of seasonal items by month, where number of sales would be fewer outside of the relevant season.

To simulate these conditions, we created a "Skewed" condition by modifying the dataset so that 22% of all records fell into one group, and the remaining data items were evenly distributed across the remaining 39 groups. It now takes Wander Join over 50,000 samples to achieve the same .05 relative error across all groups, nearly double the number of samples for an evenly distributed dataset. The group with 22% of the data is sampled more often than the other groups and reaches .05 relative error first. However, it is still sampled while the other groups converge, reducing the rate of convergence for the remaining groups, but also improving the relative error of the dominate group. The group with 22% of the data reaches .01 relative error while the other groups reach .05. This is 5 times higher in error of the estimate and a user cannot accurately compare the group estimates.

4 Limitations of Wander Join

The above experiments demonstrate both the potential of Wander Join and its limitations. Wander Join can support fast, iterative queries that do not require pre-computation and storage. Further, it supports `JOIN`s that allow for flexible data analysis. In sum, Wander Join represents the state-of-the-art in online aggregation techniques.

However, Wander Join also has its limitations. As shown in the experiments above, in cases where the query involves highly selective filters, either through `WHERE` or `GROUP BY` clauses, Wander Join's performance suffers. In some cases, it might be more beneficial to perform the full `JOIN` instead of using Wander Join's sampling approach. Unfortunately, since these highly selective filters are common in visualization applications, improving Wander Join is necessary before it can be readily adopted by visualization researchers and practitioners.

5 Load-n-Go: Wander Join for Visual Data Exploration

To address the limitations described above, we extended Wander Join to develop Load-n-Go. The key algorithmic insight in Load-n-Go is to take the `WHERE` and `GROUP BY` clauses into account when drawing samples from the database by prioritizing samples that are more likely to satisfy the filters. We achieve this by integrating the idea of importance sampling [18] into Wander Join, which allows us to prioritize samples by weighting them based on their importance to the query. Wander Join uniformly samples (the weight of each record is the same for all records), which can degrade the convergence rate under filtering and `GROUP BY` queries. By adding importance sampling, we can sample non-uniformly by changing record weights.

We apply importance sampling in two ways. For filter queries, we set the weight to 0 for records that do not pass the filters. This prevents Load-n-Go from sampling the record again in the future, thereby reducing sample failures and the overall number of samples needed to reach convergence. All other records are sampled from uniformly.

Secondly, importance sampling is used to uniformly sample from all groups in a `GROUP BY` query, resulting in all groups converging at the same rate, regardless of the number of records in each group. We weight each record based on the number of records that also fall into that record's group and start sampling from the table containing the `GROUP BY` attribute. This allows Load-n-Go to uniformly sample based on *group* instead of uniformly by *record* and all groups converge at the same rate. Now the user is not waiting for a group with low membership to converge. The following sections discuss how we applied importance sampling in more detail, as well as our evaluation of the methods and comparison to Wander Join.

5.1 Evaluating Highly Selective Queries

Filtering data is a common task of exploratory analysis [36]. Consider a retail company business analyst interested in the average a customer spends on clothing, as opposed to all the items her company sells. Her query uses a filter to limit the average calculation to only include clothing items. As seen in section 3.1.3, Wander Join's performance in this scenario can be worst than executing the full query.

Our solution is to prune out samples that do not pass the filter. By eliminating those records, we prevent sample failures and can converge with less samples (and therefore faster) than Wander Join.

5.1.1 Method

When a filtering query is issued, we set the weight of each record uniformly. As we sample, if we encounter a record that fails the filter, we set the record's weight to 0 and consider this sample a failure. Although we still have a failure in the same sense as Wander Join, the key difference from Wander Join is that by setting the weight to 0, we will guarantee that we never sample this failed record again. This is beneficial since Wander Join's sampling is with replacement; any record can be sampled multiple times, regardless of whether the record has failed the filter criteria once before. The bigger benefit is that this record is pruned from *all* paths that lead to it. Using Figure 2 as an example, we see that two possible paths to sample are a1→b1→c2 and a1→b2→c2. If Load-n-Go selects a1, then b1, then c2 and record c2 fails the filter, c2's weight is now 0 and is pruned. Now not only has a1→b1→c2 been eliminated as a possible path, but also a1→b2→c2. By pruning out the failed record, we prevent any path from sampling that record again. We also recursively prune the path when possible. For example, if a record in Table A can only `JOIN` with records that have been pruned out in Table B, then the record in Table A can be pruned out as well. We continue recursively pruning back up the path when we encounter a failure.

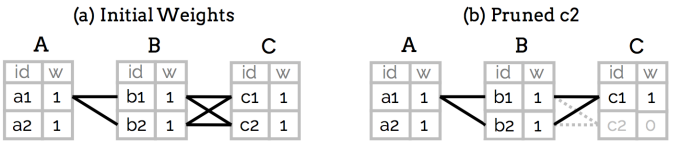

Fig. 2. (a) A JOIN graph for a 3 table JOIN. Each record starts with a weight of 1. If Load-n-Go chooses the path a1→b1→c2 and c2 fails the filter, c2 is pruned. (b) The resulting JOIN graph after pruning. Note that the path a1→b2→c2 has been pruned even though we never selected that path. Pruning out c2 prevents any path from sampling c2 again for this query.

5.1.2 Evaluation

We ran the same filtering queries as in section 3.1.3, varied the *part.p_size* and *lineitem.l_quantity* filter to achieve levels of 0% to 99% filtering and used the same 10MB TPC-H dataset.

Figure 3 captures the results of the selective filter queries. The higher the selective percentage, the larger improvement over Wander Join. For the 99% selective filter query, Load-n-Go achieved .08 relative error after 10,000 samples, while Wander Join had .22 relative error. In terms of number of samples, Wander Join needed 65,000 to reach .08 relative error. Load-n-Go is an improvement over Wander Join by reducing the sample complexity by 85% (and there by speeding up convergence by a factor of 6). Load-n-Go outperformed Wander Join at all selective filter queries and performed the same at the 0% selective filter query (since no records were filtered out).

5.2 Evaluating Group By Queries

As discussed in earlier sections, GROUP BY queries are ubiquitous in exploratory analysis. More often than not, the data rendered by a bar chart or heat map is computed as the result of a GROUP BY query over the x, or x and y, axis attributes.

Since Wander Join uniformly samples each record from the underlying table, each group's convergence rate depends on the proportion of records that belong to the group. Thus, it can take a large number of samples before the algorithm draws a record for an unpopular group. We address this issue, we use importance sampling to uniformly sample *from each group* by weighting the records in each group relative to the number of records in the group and the number of total groups.

5.2.1 Method

To ensure uniformly sampling from each group, we weight each record in the table referenced in the GROUP BY clause with this formula:

$$\omega_i = \frac{1}{\alpha\beta} \tag{3}$$

where α is the number of records that are in the same group as record i and β is number of distinct groups. The intuition is that we want to sample from each group evenly, and sample each record within each group evenly. The $\frac{1}{\alpha}$ term ensures uniform sampling of records in a given group, while $\frac{1}{\beta}$ ensures uniformly sampling from each group.

With this weighting, Load-n-Go randomly selects from the GROUP BY table first, guaranteeing even sampling from each group. The records in the next tables are weighted uniformly as before. Using Figure 1 as an example, assume Table A contains the attribute we will group on and records $a1...an$ will be weighted according to equation 3. Records in Table B and Table C will be weighted uniformly. We do not need to adjust their sampling rates since we have already guaranteed even sampling by group from reweighting records in Table A.

5.2.2 Evaluation

We ran the same GROUP BY query as in section 3.1.3 on the evenly distributed TPC-H dataset, where equal number of records fell into each group. We also tested with the same skewed dataset as in 3.1.3, so that 22% of the data fell into one group and the remaining data was evenly distributed across the remaining 39 groups.

To achieve .05 relative error across all groups, the same number of samples are needed regardless of the data distribution, since Load-n-Go samples evenly from each group, and the groups converge at the same rate. This is in contrast to Wander Join, which needed twice as many samples as Load-n-Go to reach the same relative error.

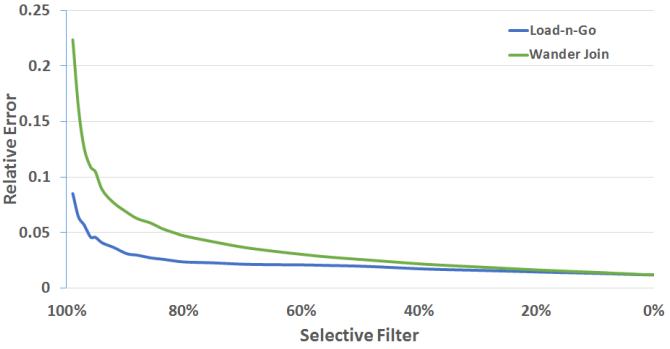

Fig. 3. The relative error at 95% CI of Wander Join and Load-n-Go after 10,000 samples over various levels of selective filtering for the TPC-H dataset (with 60,175 rows in the full JOIN).The selective filter percentage is the percent of records in the full JOIN that do not pass the filter conditions.The higher the selective filter percentage is, the slower the rate of convergence.By pruning out records that fail the filter, our relative error is reduced by nearly a factor of 3 over standard Wander Join at 99% selective filter.

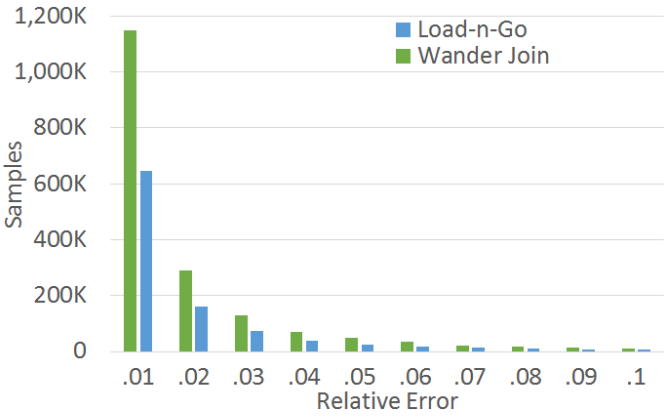

Fig. 4. The number of samples needed for Wander Join and Load-n-Go to achieve 0.1 to 0.01 relative error for all 40 groups in a GROUP BY query using the skewed dataset. Load-n-Go requires less samples than Wander Join to achieve all relative errors on a skewed dataset. Wander Join requires nearly double the number of samples as Load-n-Go to achieve relative errors 0.05 and lower.

Figure 4 shows the number of samples needed for Load-n-Go and Wander Join to reach relative errors 0.1 to 0.01 on the skewed dataset. Load-n-Go required 25% to 50% fewer samples than Wander Join to reach the same relative error.

6 Conclusion and Future Work

We present Load-n-Go, a progressive visualization system that extends online aggregation to enable interactive data exploration. Load-n-Go improves on the latest online aggregation algorithm, Wander Join, by using importance sampling. Importance sampling allows Load-n-Go to converge much faster on highly selective WHERE queries and converge uniformly on all groups in GROUP BY queries, regardless of data membership distribution.

We showed that Load-n-Go outperforms Wander Join on filtering queries and performs equally as well on non-filtering queries. We also showed that Load-n-Go requires up to 50% fewer samples than Wander Join to converge on all groups in a GROUP BY query.

Through Load-n-Go's evaluation, we identified multiple opportunities for future work. Being able to preprune out records that do not pass query filters would improve Load-n-Go's convergence rate even further. We'd also like to explore importance sampling as applied to rare or extreme values, so they are not missed in the sampling process.Load-n-Go has shown to be an improvement over current online aggregation methods for visualization tasks and through future work can be an even more capable system.

REFERENCES

[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *ACM Sigmod Record*, volume 28, pages 574–576. ACM, 1999.

[2] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM, 2013.

[3] D. Alabi and E. Wu. Pfunk-h: approximate query processing using perceptual models. In *HILDA@ SIGMOD*, page 10, 2016.

[4] L. Battle, R. Chang, and M. Stonebraker. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1363–1375. ACM, 2016.

[5] J.-H. Böse, A. Andrzejak, and M. Högqvist. Beyond online aggregation: parallel and incremental data mining with online map-reduce. In *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud*, page 3. ACM, 2010.

[6] U. Cetintemel, M. Cherniack, J. DeBrabant, Y. Diao, K. Dimitriadou, A. Kalinin, O. Papaemmanouil, and S. B. Zdonik. Query steering for interactive data exploration. In *CIDR*, 2013.

[7] S.-M. Chan, L. Xiao, J. Gerth, and P. Hanrahan. Maintaining interactivity while exploring massive time series. In *Visual Analytics Science and Technology,f 2008. VAST'08. IEEE Symposium on*, pages 59–66. IEEE, 2008.

[8] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot, K. Elmeleegy, and R. Sears. Online aggregation and continuous query support in mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1115–1118. ACM, 2010.

[9] B. Ding, S. Huang, S. Chaudhuri, K. Chakrabarti, and C. Wang. Sample+ seek: Approximating aggregates with distribution precision guarantee. In *Proceedings of the 2016 International Conference on Management of Data*, pages 679–694. ACM, 2016.

[10] P. R. Doshi, E. Geraldine, G. Rosario, E. Rundensteiner, and M. Ward. A strategy selection framework for adaptive prefetching in data visualization. In *Scientific and Statistical Database Management, 2003. 15th International Conference on*, pages 107–116. IEEE, 2003.

[11] J.-D. Fekete. Progressivis: a toolkit for steerable progressive analytics and visualization. In *1st Workshop on Data Systems for Interactive Analysis*, page 5, 2015.

[12] J.-D. Fekete and R. Primet. Progressive analytics: A computation paradigm for exploratory data analysis. *arXiv preprint arXiv:1607.05162*, 2016.

[13] D. Fisher. Incremental, approximate database queries and uncertainty for exploratory visualization. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 73–80. IEEE, 2011.

[14] D. Fisher, S. M. Drucker, and A. C. König. Exploratory visualization involving incremental, approximate database queries and uncertainty. *IEEE computer graphics and applications*, 32(4):55–62, 2012.

[15] D. Fisher, I. Popov, S. Drucker, et al. Trust me, i'm partially right: incremental visualization lets analysts explore large datasets faster. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1673–1682. ACM, 2012.

[16] P. Godfrey, J. Gryz, and P. Lasek. Interactive visualization of large data sets. *IEEE Transactions on Knowledge and Data Engineering*, 28(8):2142–2157, 2016.

[17] P. J. Haas and J. M. Hellerstein. Ripple joins for online aggregation. *ACM SIGMOD Record*, 28(2):287–298, 1999.

[18] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[19] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *ACM SIGMOD Record*, volume 26, pages 171–182. ACM, 1997.

[20] D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association*, 47(260):663–685, 1952.

[21] J.-F. Im, F. G. Villegas, and M. J. McGuffin. Visreduce: Fast and responsive incremental information visualization of large datasets. In *Big Data, 2013 IEEE International Conference on*, pages 25–32. IEEE, 2013.

[22] V. Kalavri, V. Brundza, and V. Vlassov. Block sampling: Efficient accurate online aggregation in mapreduce. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 250–257. IEEE, 2013.

[23] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi. Distributed and interactive cube exploration. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 472–483. IEEE, 2014.

[24] A. Kemper and T. Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 195–206. IEEE, 2011.

[25] A. Kim, E. Blais, A. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld. Rapid sampling for visualizations with ordering guarantees. *Proceedings of the VLDB Endowment*, 8(5):521–532, 2015.

[26] F. Li, B. Wu, K. Yi, and Z. Zhao. Wander join: Online aggregation via random walks. In *Proceedings of the 2016 International Conference on Management of Data*, pages 615–629. ACM, 2016.

[27] X. Li, J. Han, Z. Yin, J.-G. Lee, and Y. Sun. Sampling cube: a framework for statistical olap over sampling data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 779–790. ACM, 2008.

[28] L. Lins, J. T. Klosowski, and C. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2456–2465, 2013.

[29] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, volume 32, pages 421–430. Wiley Online Library, 2013.

[30] D. Moritz, D. Fisher, B. Ding, and C. Wang. Trust, but verify: Optimistic visualizations of approximate queries for exploring big data. 2017.

[31] C. A. Pahins, S. A. Stephens, C. Scheidegger, and J. L. Comba. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):671–680, 2017.

[32] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large mapreduce jobs. *Proc. VLDB Endow*, 4(11):1135–1145, 2011.

[33] N. Pezzotti, B. Lelieveldt, L. van der Maaten, T. Hollt, E. Eisemann, and A. Vilanova. Approximated and user steerable tsne for progressive visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 2016.

[34] R. Rosenbaum and H. Schumann. Progressive refinement: more than a means to overcome limited bandwidth. In *IS&T/SPIE Electronic Imaging*, pages 72430I–72430I. International Society for Optics and Photonics, 2009.

[35] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34. ACM, 1979.

[36] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. IEEE, 1996.

[37] C. D. Stolper, A. Perer, and D. Gotz. Progressive visual analytics: User-driven visual exploration of in-progress analytics. *IEEE transactions on visualization and computer graphics*, 20(12):1653–1662, 2014.

[38] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002.

[39] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, et al. C-store: a column-oriented dbms. In *Proceedings of the 31st international conference on Very large data bases*, pages 553–564. VLDB Endowment, 2005.

[40] Z. Wang, N. Ferreira, Y. Wei, A. S. Bhaskar, and C. Scheidegger. Gaussian cubes: Real-time modeling for visual exploration of large multidimensional datasets. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):681–690, 2017.

[41] E. Wu, L. Battle, and S. R. Madden. The case for data visualization management systems: vision paper. *Proceedings of the VLDB Endowment*, 7(10):903–906, 2014.

[42] E. Wu, F. Psallidas, Z. Miao, H. Zhang, L. Rettig, Y. Wu, and T. Sellam. Combining design and performance in a data visualization management system.

[43] E. Zgraggen, A. Galakatos, A. Crotty, J.-D. Fekete, and T. Kraska. How progressive visualizations affect exploratory analysis. *IEEE Transactions on Visualization and Computer Graphics*, 2016.