Algorithm 2 Algorithm for the *Post-Filtering* strategy.

Require: WorkloadQuery
Require: ProvenanceQuery
(ctxSet) = ExecuteQuery(ProvenanceQuery)
(results, polynomial) = ExecuteQuery(WorkloadQuery) {independent execution of ProvenanceQuery and WorkloadQuery}
for all results **do**
  if (polynomial[result].ContextValues ⊈ ctxSet) **then**
    remove result
  else
    keep result
  end if
end for

Algorithm 3 Algorithm for the *Rewriting* strategy.

Require: query: workload query
Require: ctxSet: context values; results of provenance query
tuples =q.getPhysicalPlan (FROM materializedTuples for materializes scenario)
for all tuples **do**
  for all entities **do**
    if (entity.ContextValues ⊈ ctxSet) **then**
      nextEntity
    else
      inspect entity
    end if
  end for
end for

Algorithm 4 Algorithm for the *Pre-Filtering* strategy.

Require: query: workload query
Require: ctxSet: context values; results of provenance query
tuples =q.getPhysicalPlan
for all tuples **do**
  for all ctxSet **do**
    ctxTuples = getTuplesFromProvIdx(ctx)
    if (tuple ⊈ ctxTuples) **then**
      nextTuple
    end if
  end for
  for all entities **do**
    if (entity.ContextValues ⊈ ctxSet) **then**
      nextEntity
    else
      inspect entity
    end if
  end for
end for

of this strategy requires the introduction of an additional data structure at the core of the system, and the adjustment of the query execution process in order to use it.

5. STORAGE MODEL & INDEXING

We implemented all the provenance-enabled query execution strategies introduced in Section 4 in TripleProv, our own triplestore supporting different storage models to handle provenance data. Both TripleProv[7] and the extensions implemented for this paper[8] are available online. In the following, we briefly present the implementation of the provenance-oriented data structures and indices we used to evaluate the query execution strategies described above. We note that it would be possible to implement our strategies in other systems, using the same techniques. The effort to do so, however, is beyond the scope of the paper.

[7] `http://exascale.info/tripleprov`
[8] `http://exascale.info/provqueries`

Algorithm 5 Algorithm for the *Partial Materialization* strategy.

Require: query: workload query
Require: ctxSet: context values; results of provenance query
Require: collocatedTuples: collection of hash values of tuples related to the result of the provenance query (ctxSet)
tuples =q.getPhysicalPlan
for all tuples **do**
  if (tuple ⊈ collocatedTuples) **then**
    nextTuple
  end if
  for all entities **do**
    if (entity.ContextValues ⊈ ctxSet) **then**
      nextEntity
    else
      inspect entity
    end if
  end for
end for

5.1 Provenance Storage Model

We use the most basic storage structure of Diplodocus[RDF] [34] and TripleProv [33] in the following: 1-scope RDF molecules [34], which collocate objects related to a given subject and which are equivalent to property tables. In that sense, any *tuple* we consider is composed of a subject, and a series of predicate and object related to that subject.

TripleProv supports different models to store provenance information. We compared those models in [33]. For this work, we consider the "SLPO" storage model [33], which collocates the context values with the predicate-object pairs, and which offers good overall performance in practice. This avoids the duplication of the same context value, while at the same time collocating all data about a given subject in one structure. The resulting storage model is illustrated in Figure 3. In the rest of this section, we briefly introduce the secondary storage structures we implemented to support the query execution strategies of Section 4.

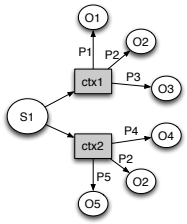

Figure 3: Our storage model for collocating context values (ctx) with predicates and objects (PO) inside an RDF molecule.

5.2 Provenance Index

Our base system supports a number of vertical and horizontal data collocation structures. Here, we propose one more way to collocate molecules, based on the context values. This gives us the possibility to prune molecules during query execution as explained above. Figure 4 illustrates this index, which boils down, in our implementation, to lists of collocated molecule identifiers, indexed by a hash-table whose keys are the context values the triples stored in the molecules belong to. We note that a given molecule can

appear multiple times in this index. This index is updated upfront, e.g., at loading time.

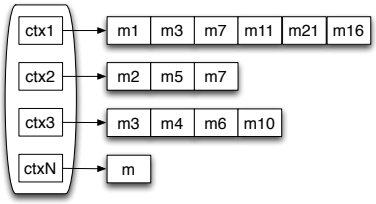

Figure 4: Provenance-driven indexing schema

5.3 Provenance-Driven Full Materialization

To support the provenance-driven materialization scheme introduced in Section 4.3, we implemented some basic view mechanisms in TripleProv. These mechanisms allow us to project, materialize and utilize as a secondary structure the portions of the molecules that are following the provenance specification (see Figure 5 for a simple illustration.)

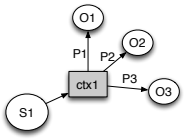

Figure 5: The molecule after materialization, driven by a provenance query returning only one context value (ctx1).

5.4 Adaptive Partial Materialization

Finally, we implement a new, dedicated structure for the adaptive materialization strategy. In that case, we collocate all molecule identifiers that are following the provenance specification (i.e., that contain *at least* one context value compatible with the provenance query). We explored several options for this structure and in the end implemented it through a *hashset*, which yields constant time performance to insert molecules when executing the provenance query and to query for molecules when executing workload queries.

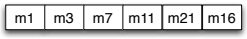

Figure 6: Set of molecules which contain at least some data related to a provenance query.

6. EXPERIMENTS

To empirically evaluate the query execution strategies discussed above in Section 4.3, we implemented them all in TripleProv. In the following, we experimentally compare a baseline version of our system that does not support provenance queries to our five strategies executing provenance-enabled queries. We perform the evaluation on two different datasets and workloads.

Within TripleProv, queries are specified as triple patterns using a high-level declarative API that offers similar functionality to SPARQL.[9] The queries are then encoded into a logical plan (a tree of operators), which is then optimized into a physical query plan as in any standard database system. The system supports all basic SPARQL operations, including "UNION" and "OPTIONAL"; at this point, it does not support "FILTER", however.

6.1 Implementations Considered

Our goal is to understand the various tradeoffs of the query execution strategies we proposed in Section 4.3 and to assess the performance penalty (or eventual speed-up) caused by provenance queries. We use the following abbreviations to refer to the different implementations we compare:

TripleProv: the vanilla version of [33], without provenance queries; this version stores provenance data, tracks the lineage of the results, and generates provenance polynomials, but does not support provenance queries;

Post-Filtering: implements our post-filtering approach; after a workload query gets executed, its results are filtered based on the results from the provenance query;

Rewriting: our query execution strategy based on query rewriting; it rewrites the workload query by adding provenance constraints in order to filter out the results;

Full Materialization: creates a materialized view based on the provenance query, and executes the workload queries over that view;

Pre-Filtering: uses a dedicated provenance index to pre-filter tuples during query execution;

Adaptive Materialization: implements a provenance-driven data co-location scheme to collocate molecule ids that are relevant given the provenance query.

6.2 Experimental Environment

Hardware Platform: All experiments were run on a HP ProLiant DL385 G7 server with an AMD Opteron Processor 6180 SE (24 cores, 2 chips, 12 cores/chip), 64GB of DDR3 RAM, running Ubuntu 12.04.3 LTS (Precise Pangolin). All data were stored on a recent 3 TB Serial ATA disk.

Datasets: We used two different datasets for our experiments: the Billion Triples Challenge (BTC)[10] and the Web Data Commons (WDC)[11] [26]. Both datasets are collections of RDF data gathered from the Web. They represent two very different kinds of RDF data. The Billion Triple Challenge dataset was created based on datasets provided by Falcon-S, Sindice, Swoogle, SWSE, and Watson using the MultiCrawler/SWSE framework. The Web Data Commons project extracts all Microformat, Microdata and RDFa data from the Common Crawl Web corpus and provides the extracted data for download in the form of RDF-quads or CSV-tables for common entity types (e.g., products, organizations, locations, etc.).

Both datasets represent typical collections of data gathered from multiple and heterogeneous online sources, hence applying some provenance query on them seems to precisely address the problem we focus on. We consider around 40 million triples for each dataset (around 10GB). To sample the data, we first pre-selected quadruples satisfying the set of considered workload and provenance queries. Then, we randomly sampled additional data up to 10GB.

For both datasets, we added provenance specific triples (184 for WDC and 360 for BTC) so that the provenance queries we use for all experiments do not modify the result sets of the workload queries, i.e., the workload query results are always the same. We decided to implement this

[9]We note that our current system does not parse full SPARQL queries at this stage. Adapting a SPARQL parser is currently in progress.

[10]`http://km.aifb.kit.edu/projects/btc-2009/`

[11]`http://webdatacommons.org/`

to remove a potential bias when comparing the strategies and the vanilla version of the system (in this way, in all cases all queries have exactly the same input and output). We note that this scenario represents in fact a worst-case scenario for our provenance-enabled approaches, since the provenance query gets executed but does not filter out any result. Therefore, we also performed experiments on the original data (see Section 6.3.4), where we use the dataset as is and where the provenance query modifies the output of the workload queries.

Workloads: We consider two different workloads. For BTC, we use eight existing queries originally proposed in [27]. In addition, we added two queries with UNION and OPTIONAL clauses, which we thought were missing in the original set of queries. Based on the queries used for the BTC dataset, we wrote 7 new queries for the WDC dataset, encompassing different kinds of typical query patterns for RDF, including star-queries of different sizes and up to 5 joins, object-object joins, object-subject joins, and triangular joins. We also included two queries with UNION and OPTIONAL clauses. In addition, for each workload we prepared a complex provenance query, which is conceptually similar to those presented in Section 3.

The datasets, query workloads and provenance-queries presented above are all are available online[12].

Experimental Methodology: As is typical for benchmarking database systems (e.g., for tpc-x[13] or our own OLTP-Benchmark [11]), we include a warm-up phase before measuring the execution time of the queries in order to measure query execution times in a steady-state mode. We first run all the queries in sequence once to warm-up the system, and then repeat the process ten times (i.e., we run 11 query batches for each variant we benchmark, each containing all the queries we consider in sequence). We report the average execution time of the last 10 runs for each query. In addition, we avoided the artifacts of connecting from the client to the server, of initializing the database from files, and of printing results; we measured instead the query execution times inside the database system only.

6.3 Results

In this section, we present the results of the empirical evaluation. We note that our original RDF back-end Diplodocus (the system TripleProv extends) has already been compared to a number of other well-known triple stores(see [34] and [10]). We refer the reader to those previous papers for a comparison to non-provenance-enabled triple stores. We have also performed an evaluation of TripleProv and different physical models for storing provenance information in [33]. In this paper, we focus on a different topic and discuss results for *Provenance-Enabled Queries*. Figure 7 reports the query execution times for the BTC dataset, while Figure 8 shows similar results for the WDC dataset. We analyze those results below.

6.3.1 Datasets Analysis

To better understand the influence of provenance queries on performance, we start by taking a look at the dataset, provenance distribution, workload, cardinality of intermediate results, number of molecules inspected, and number

[12] `http://exascale.info/provqueries`
[13] *http://www.tpc.org/*

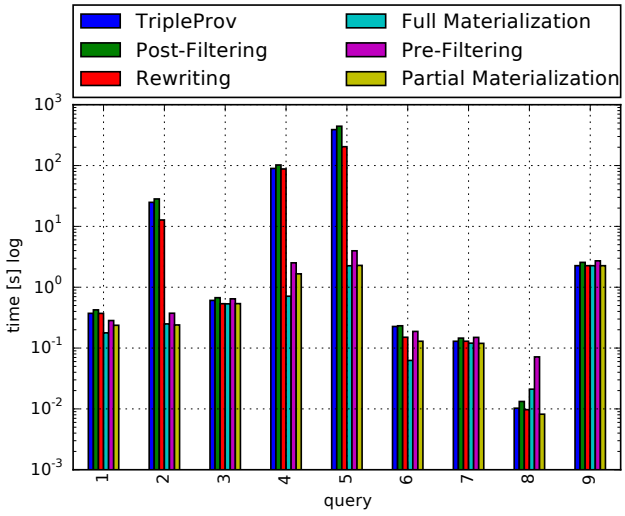

Figure 7: Query execution times for the BTC dataset (logarithmic scale)

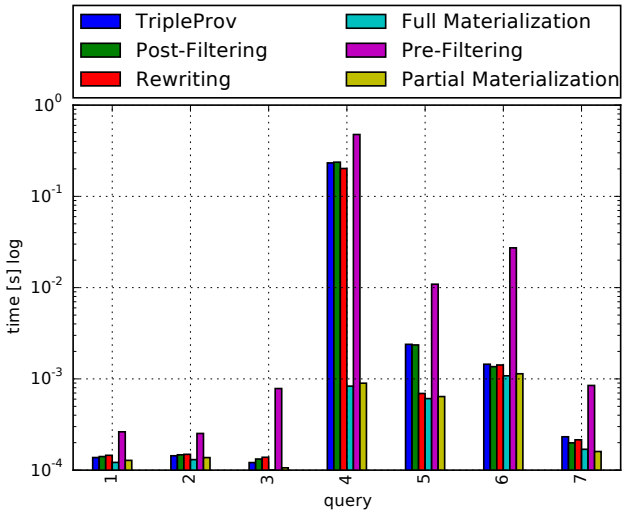

Figure 8: Query execution times for the WDC dataset (logarithmic scale).

of basic operations for all query execution strategies. The analysis detailed below was done for the BTC dataset and workload.

First, we analyze the distribution of context values among triples. There are 6'819'826 unique context values in the dataset. Figure 9 shows the distribution of the number of triples given the context values (i.e., how many context values refer to how many triples). We observe that there are only a handful of context values that are widespread (left-hand side of the figure) and that the vast majority of the context values are highly selective. On average, each context value is related to about 5.8 triples. Collocating data inside molecules further increases the selectivity of the context values, we have on average 2.3 molecules per context value then. We leverage those properties during query execution, as some of our strategies prune molecules early in the query plan based on their context values.

6.3.2 Discussion

Our implementations supporting provenance-enabled queries overall outperform the vanilla TripleProv. This

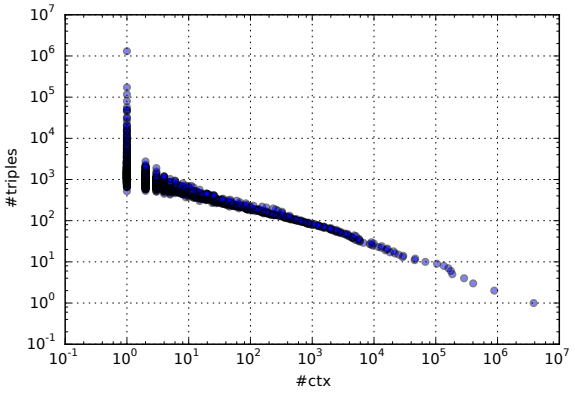

Figure 9: Distribution of number of triples for number of context values for the BTC dataset.

is unsurprising, since as we showed before the selectivity of provenance data in the datasets allows us to avoid unnecessary operations on tuples which do not add to the result.

The *Full Materialization* strategy, where we pre-materialize all relevant subsets of the molecules, makes the query execution on average 44 times faster than the vanilla version for the BTC dataset. The speedup ranges from a few percents to more than 200x (queries 2 and 5 of BTC) over the vanilla version. The price for the performance improvement is the time we have to spend to materialize molecules, in our experiments for the BTC it was 95 seconds (the time increases with data size), which can however be amortized by executing enough workload queries (see Section 6.4). This strategy consumed about 2% more memory for handling the materialized data.

The *Pre-Filtering* strategy performs on average 23 times faster than the vanilla version for the BTC dataset, while the *Adaptive Partial Materialization* strategy performs on average 35 times faster for the BTC dataset. The advantage over the *Full Materialization* strategy is that for *Adaptive Partial Materialization*, the time to execute a provenance query and materialize data is 475 times lower and takes only 0.2 second.

The *Query Rewriting* strategy performs significantly slower than the strategies mentioned above for the BTC dataset, since here we have to perform additional checking over provenance data for each query. However, even in this case, for some queries we can observe some performance improvement over the vanilla version of the system; when the provenance query significantly limits the number of tuples inspected during query execution (see Section 4), we can compensate the time taken by additional checks to improve the overall query execution time—see queries 2 and 5 for BTC. Those queries can be executed up to 95% faster than the vanilla version as they require the highest number of tuple inspections, which can significantly limit other strategies (see Section 6.3.3).

We note that the *Post-Filtering* strategy performs in all cases slightly worse than TripleProv (on average 12%), which is expected since there is no early pruning of tuples; queries are executed in the same way as in TripleProv, and in addition the post-processing phase takes place to filter the results set.

For the WDC dataset we have significantly higher cardinality of context values set (10 times more elements), which results in significantly worse performance for *Pre-Filtering*, since this strategy performs a loop over the set of context values. The provenance overhead here is not compensated on workload query execution since they are already quite fast (below 10^{-2} second for most cases) for this dataset. For this scenario the time consumed for *Full Materialization* was 60 seconds while it took only 2ms for *Adaptive Partial Materialization*. The *Adaptive Partial Materialization* strategy outperforms other strategies even more clearly on the WDC dataset.

The WDC workload shows an even higher predominance of the *Adaptive Partial Materialization* strategy over other strategies.

6.3.3 Query Performance Analysis

We now examine the reasons behind the performance differences for the different strategies, focusing on the BTC dataset. Thanks to materialization and collocation, we limit the number of molecule look-ups we require to answer the workload queries. The tables below explain the reasons behind the difference in performance. We analyze the number of inspected molecules, the number of molecules after filtering by provenance, the cardinality of intermediate results, and the number of context values used to answer the query:

#r - number of results
#m - total number of molecules used to answer the query, before checking against context values
#mf - total number of molecules after pruning with provenance data
#prov - total number of provenance context values used to answer the query (to generate a polynomial)
#im - intermediate number of molecules used to answer the query, before checking against context values
#imf - intermediate number of molecules after pruning with provenance data
#i - number of intermediate results, used to perform joins
#ec - number of basic operation executed on statements containing only constraints in a query
#er - number of basic operation executed on statements containing projections in a query

The total number of executed basic operations (*#bos*) equals $\#ec + \#er$.

We prepared the provenance query to ensure that the results for all variants are constant, therefore we avoid the bias of having different result sets.

query #	#r	#m	#mf	#prov	#im	#imf	#i	#ec	#er
1	2	4	4	2	0	0	0	84039	470
2	9	203	203	4	0	0	0	3698911	8392
3	13	32	32	7	0	0	0	18537	5580
4	5	1335	1335	5	1	1	1	44941143	4048
5	5	3054	3053	8	3052	3052	3	79050305	37040
6	2	137	133	6	136	132	374	22110	8365
7	2	20	6	5	2	2	18	438	7239
8	237	267	251	287	0	0	0	752	0
9	17	32	32	8	0	0	0	18537	101420

Table 1: Query execution analysis for TripleProv and the *Post-Filtering* strategy.

Table 1 shows the baseline statistics for the vanilla version, TripleProv.

Table 2 give statistics for the *Rewriting*. We observe at this level already that we inspect data from on average 50x fewer molecules, which results on average in a 30% boost in performance. However, executing the provenance query also has its price, which balances this gain in performance for simpler queries (e.g., 7-9).

query #	#r	#m	#mf	#prov	#im	#imf	#i	#ec	#er
1	2	4	2	2	0	0	0	5438	470
2	9	203	1	4	0	0	0	832980	6176
3	13	32	32	6	0	0	0	9715	3990
4	5	1335	22	5	1	1	1	1666409	3304
5	5	3054	18	8	3052	17	3	2163812	8008
6	2	137	98	6	136	97	6	13434	5506
7	2	20	2	5	2	1	18	399	7211
8	237	267	237	287	0	0	0	580	0
9	17	32	32	7	0	0	0	9715	52220

Table 2: Query execution analysis for the *Rewriting* strategy.

query #	#r	#m	#mf	#prov	#im	#imf	#i	#ec	#er
1	2	1	1	2	0	0	0	4660	466
2	9	1	1	4	0	0	0	832426	4144
3	13	31	31	6	0	0	0	2801	2826
4	5	8	8	5	1	1	1	87716	2386
5	5	16	15	8	14	14	3	1865699	4662
6	2	102	98	6	101	97	6	10279	4513
7	2	15	2	5	1	1	14	284	7102
8	237	237	237	287	0	0	0	435	0
9	17	31	31	7	0	0	0	2801	5114

Table 3: Query execution analysis for the *Full Materialization* strategy.

Table 3 gives statistics for our second variant (*Full Materialization*). The total number of molecules initially available is in this case reduced by 22x. Thanks to this, the total number of molecules used to answer the query ('#m') decreases on average 63x; we also reduce the number of molecules inspected after pruning with provenance data ('#mf') by 33% compared to the baseline version. This results in a performance improvement of 29x on average. For some queries (3, 7 and 9), the number of inspected molecules remains almost unchanged, since the workload query itself is very selective and since there is no room for further pruning molecules before inspecting them. Those queries perform similarly as for the baseline version. For queries 2, 4, and 5, we observe that the reduction in terms of the number of molecules used is 200x, 166x, and 190x, respectively, which significantly impacts the final performance. The price to pay for these impressive speedups is the time spent on the upfront materialization, which was 95 seconds for the dataset considered.

query #	#r	#m	#mf	#prov	#im	#imf	#i	#ec	#er
1	2	2	2	2	0	0	0	5436	470
2	9	1	1	4	0	0	0	832680	6176
3	13	32	32	6	0	0	0	9715	3990
4	5	22	22	5	1	1	1	1663384	3304
5	5	19	18	8	17	17	3	2159510	8008
6	2	102	98	6	101	97	6	13353	5506
7	2	15	2	5	1	1	18	393	7211
8	237	237	237	287	0	0	0	537	0
9	17	32	32	7	0	0	0	9715	52220

Table 4: Query execution analysis for the *Pre-Filtering* and *Adaptive Partial Materialization* strategies.

Table 4 gives statistics for our last two implementations using *Pre-Filtering* and *Adaptive Partial Materialization*. The statistics are similar for both cases (though the structures used to answer the queries and the query execution strategies vary, as explained in Sections 4 and 5). Here the cardinality of the molecule sets remains unchanged with respect to the vanilla version, and the total number of molecules used to answer the query is identical to molecules after provenance filtering for the naive version, but all molecules we inspect contain data related to the provenance query ('#m' and '#mf' are equal for each query). In fact, we inspect a number of molecules similar to *Full Materialization*, which yields performance of a similar level, on average 14x (*Pre-Filtering*) and 22x (*Adaptive Partial Materialization*) faster than the *Rewriting* strategy. The cost of materialization for *Adaptive Partial Materialization* is much lower than for *Full Materialization*, however, as the strategy only requires 0.2 extra second in order to dynamically co-locate molecules containing data relevant for the provenance query.

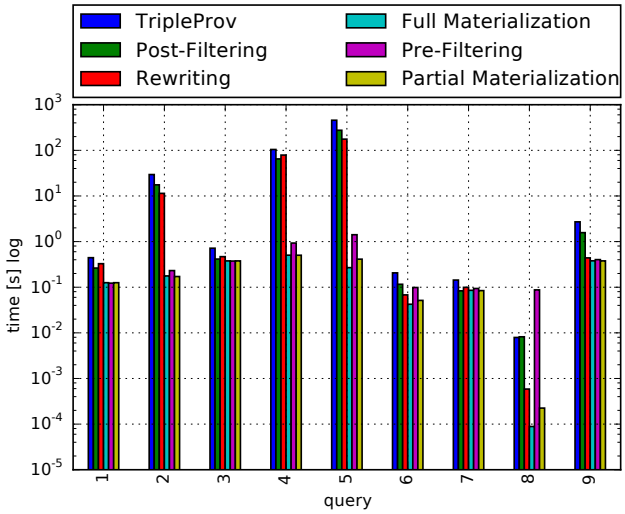

Figure 10: Query execution times for the BTC dataset (logarithmic scale), Representative Scenario.

6.3.4 Representative Scenario

As we mentioned above, our experiments so far aimed at comparing the execution times for different strategies fairly, thus we prepared an experimental scenario where the final output remains unchanged for all implementations (including vanilla TripleProv). In this section, we present a microbenchmark depicting a representative scenario run on the original BTC dataset (without any triples added), where the output changes due to constraints imposed on the workload by the provenance query. This dataset is also available online on the project web page.

Table 5 shows the corresponding query execution analysis. The number of results is in this case smaller for many queries as results are filtered out based on their context values.

query #	#r	#m	#mf	#prov	#im	#imf	#i	#ec	#er
1	2	1	1	1	0	0	0	2166	222
2	8	1	1	2	0	0	0	604489	5064
3	10	4	4	4	0	0	0	2002	2970
4	5	8	8	3	1	1	1	82609	2768
5	3	5	5	4	4	4	1	1381357	6364
6	1	4	4	4	3	3	1	5601	2523
7	1	15	2	3	1	1	18	297	4079
8	5	5	5	4	0	0	0	5	0
9	10	4	4	4	0	0	0	2002	2970

Table 5: Query execution analysis for the Pre-Filtering and Partial Materialization strategies for the Representative Scenario.

Figure 10 shows query performance results for the original BTC dataset.

As shown on Figure 10, the performance gains for all provenance-enabled strategies are higher in this more realistic scenario where we did not modify the original data. The speedup is caused by the smaller number of *basic operations* (#*ec*+#*er*) executed, which results from fewer intermediate results. For queries for which the results remain the same (2 and 4), the improvement is directly related to the smaller number of *basic operations* performed caused by the limited number of context values resulting from the provenance query.

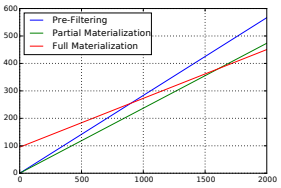

Figure 11: Cumulative query execution time including time of materialization for 2'000 repetitions of query 1 for BTC.

6.4 End-to-End Workload Optimization

Having devise several query execution strategies, it is interesting to understand which ones perform better under what circumstances. Specifically, when it pays off to use a strategy which has a higher cost for executing the provenance query and when this is not beneficial. Ideally, the time consumed on the execution of the provenance query (including some potential pre-materialization) should be compensated when executing the workload queries. Let i and j denote two different query execution strategies and P and W denote the time taken to execute the provenance and the workload queries, respectively. If: $P_i + W_i < P_j + W_j$, then strategy i should be chosen since it yields an overall lower cost for running the entire provenance-enabled workload.

As an illustration, Figure 11 shows the cumulative query execution time for query 1 of BTC including the time overhead for the provenance query execution and data materialization. We observe that the *Partial Materialization* strategy compensates the overhead of running the provenance query and of materialization after a few repetitions of the query already, compared with the *Pre-Filtering*, which has a lower cost from a provenance query execution perspective, but which executes workload queries slower. For the case of *Full Materialization*, which has a significantly higher materialization overhead, it takes about 900 workload query repetitions to amortize the cost of running the provenance query and pre-materializing data in order to beat the *Pre-Filtering* strategy. The *Full Materialization* strategy outperforms the *Partial Materialization* strategy only after more than 1'500 repetitions of the query.

In the end, the optimal strategy depends on the data, on the exact mixture of (provenance and workload) queries, and of their frequencies. Given those three parameters, one can pick the optimal execution strategy using several techniques. If the provenance and the workload queries are known in advance or do not vary much, one can run a sample of the queries using different strategies (similarly to what we did above) and pick the best-performing one. If the queries vary a lot, then one has to resort to an approximate model of query execution in order to pick the best strategy, as it is customary in traditional query optimization. Different models can by used in this context, like the very detailed main-memory model we proposed in [17], or the system-agnostic model recently proposed in [19].

As observed above, however, the performance of our various strategies are strongly correlated (with a correlation coefficient of 95%) to the number of basic operations (e.g., molecule look-ups) performed—at least as run in our system. Hence, we propose a simple though effective model in our context based on this observation. We fit a model based on experimental data giving the time to execute a varying number of basic operations. In our setting, the best model turns out to be $e^{(a*\ln bos+b)}$ (the logarithm comes from the cost of preparing a query, such as translating strings into identifiers and building the query plan, which gets amortized with a higher number of subsequent basic operations). Figure 12 shows the performance of this model in practice. Using this model and statistics about the predicates in the queries, we can successfully predict the winning strategy, i.e., *Partial Materialization* for the scenarios discussed above.

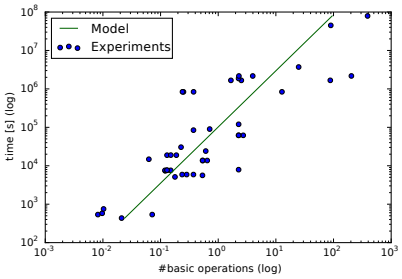

Figure 12: Query execution time vs. number of basic operations from experimental results and for our model, where the model parameters a and b were fit to 0.85 and -9.85, respectively.

7. CONCLUSIONS

In this paper, we considered the following research question: "What is the most effective query execution strategy for provenance-enabled queries"? In order to answer the research question above, this paper made the following contributions: a characterization of provenance-enabled queries, a description of five different query execution strategies, an implementation of these strategies in TripleProv, as well as a detailed performance evaluation.

The ultimate answer to this question depends on the exact data and queries used, though based on our experimental analysis above, we believe that an adaptive materialization strategy provides the best trade-off for running provenance-enabled queries over Web Data in general. Our empirical results show that this strategy performs best when taking into account the costs of materialization, both on Web Data Commons and on Billion Triple Challenge data. A key reason for this result is the selectivity of provenance on the Web of Data. Hence, by leveraging knowledge of provenance, one can execute many types of queries roughly 30x faster than a baseline store.

Building on the results of this work, we see a number of avenues of future work. The investigation of how provenance can be used to improve performance within data management systems. The development of an analytics environment which allows users to adjust provenance without changing the workload queries. Finally, the diffusion of these results to further settings and systems.

Acknowledgements

This work was supported by the Swiss National Science Foundation under grant number PP00P2_128459 and by the Dutch national program COMMIT.

References

[1] B. Arab, D. Gawlick, V. Radhakrishnan, H. Guo, and B. Glavic. A generic provenance middleware for queries, updates, and transactions. In *6th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2014)*, Cologne, June 2014. USENIX Association.

[2] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. *SIGMOD Rec.*, 29(2):261–272, May 2000.

[3] C. R. Batchelor, C. Y. A. Brenninkmeijer, C. Chichester, M. Davies, D. Digles, I. Dunlop, C. T. A. Evelo, A. Gaulton, C. A. Goble, A. J. G. Gray, P. T. Groth, L. Harland, K. Karapetyan, A. Loizou, J. P. Overington, S. Pettifer, J. Steele, R. Stevens, V. Tkachenko, A. Waagmeester, A. J. Williams, and E. L. Willighagen. Scientific lenses to support multiple views over linked chemistry data. In *ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, pages 98–113, Oct. 2014.

[4] O. Biton, S. Cohen-Boulakia, and S. B. Davidson. Zoom*userviews: Querying relevant provenance in workflow systems. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 1366–1369. VLDB Endowment, 2007.

[5] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 613–622, New York, NY, USA, 2005. ACM.

[6] A. Chebotko, S. Lu, X. Fei, and F. Fotouhi. Rdfprov: A relational rdf store for querying and managing scientific workflow provenance. *Data Knowl. Eng.*, 69(8):836–865, Aug. 2010.

[7] C. Chichester, P. Gaudet, O. Karch, P. Groth, L. Lane, A. Bairoch, B. Mons, and A. Loizou. Querying neXtProt nanopublications and their value for insights on sequence variants and tissue expression. *Web Semantics: Science, Services and Agents on the World Wide Web*, (0):–, 2014.

[8] R. L. Cole and G. Graefe. Optimization of dynamic query evaluation plans. *SIGMOD Rec.*, 23(2):150–160, May 1994.

[9] C. V. Damásio, A. Analyti, and G. Antoniou. Provenance for SPARQL queries. In *Proceedings of the 11th international conference on The Semantic Web - Volume Part I*, ISWC'12, pages 625–640, Berlin, Heidelberg, 2012. Springer-Verlag.

[10] G. Demartini, I. Enchev, M. Wylot, J. Gapany, and P. Cudre-Mauroux. BowlognaBenchâĂŤBenchmarking RDF Analytics. In K. Aberer, E. Damiani, and T. Dillon, editors, *Data-Driven Process Discovery and Analysis*, volume 116 of *Lecture Notes in Business Information Processing*, pages 82–102. Springer Berlin Heidelberg, 2012.

[11] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudré-Mauroux. Oltp-bench: An extensible testbed for benchmarking relational databases. *PVLDB*, 7(4):277–288, 2013.

[12] G. Flouris, I. Fundulaki, P. Pediaditis, Y. Theoharis, and V. Christophides. Coloring rdf triples to capture provenance. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pages 196–212, Berlin, Heidelberg, 2009. Springer-Verlag.

[13] F. Geerts, G. Karvounarakis, V. Christophides, and I. Fundulaki. Algebraic structures for capturing the provenance of sparql queries. In *Proceedings of the 16th International Conference on Database Theory*, ICDT '13, pages 153–164, New York, NY, USA, 2013. ACM.

[14] B. Glavic and G. Alonso. The perm provenance management system in action. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 1055–1058, New York, NY, USA, 2009. ACM.

[15] G. Graefe and K. Ward. Dynamic query evaluation plans. *SIGMOD Rec.*, 18(2):358–366, June 1989.

[16] P. Groth and L. Moreau (eds.). PROV-Overview. An Overview of the PROV Family of Documents. W3C Working Group Note NOTE-prov-overview-20130430, World Wide Web Consortium, Apr. 2013.

[17] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudré-Mauroux, and S. Madden. HYRISE - A main memory hybrid storage engine. *PVLDB*, 4(2):105–116, 2010.

[18] H. Halpin and J. Cheney. Dynamic Provenance for SPARQL Updates. In *The Semantic Web âĂŞ ISWC 2014*, volume 8796 of *Lecture Notes in Computer Science*, pages 425–440. Springer International Publishing, 2014.

[19] R. Hasan and F. Gandon. Predicting SPARQL query performance. In *The Semantic Web: ESWC 2014 Satellite Events - ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*, pages 222–225, 2014.

[20] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.

[21] N. Kabra and D. J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. *SIGMOD Rec.*, 27(2):106–117, June 1998.

[22] G. Karvounarakis, Z. G. Ives, and V. Tannen. Querying data provenance. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 951–962. ACM, 2010.

[23] C. A. Knoblock, P. Szekely, J. L. Ambite, S. Gupta, A. Goel, M. Muslea, K. Lerman, M. Taheriyan, and P. Mallick. Semi-automatically mapping structured sources into the semantic web. In *Proceedings of the Extended Semantic Web Conference*, Crete, Greece, 2012.

[24] S. Miles. Electronically querying for the provenance of entities. In L. Moreau and I. Foster, editors, *Provenance and Annotation of Data*, volume 4145 of *Lecture Notes in Computer Science*, pages 184–192. Springer Berlin Heidelberg, 2006.

[25] L. Moreau and G. Paul. *Provenance: An Introduction to PROV*. Morgan and Claypool, September 2013.

[26] H. Mühleisen and C. Bizer. Web data commons - extracting structured data from two large web corpora. In C. Bizer, T. Heath, T. Berners-Lee, and M. Hausenblas, editors, *LDOW*, volume 937 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[27] T. Neumann and G. Weikum. Scalable join processing on very large RDF graphs. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 627–640. ACM, 2009.

[28] K. Ng, Z. Wang, R. R. Muntz, and S. Nittel. Dynamic query re-optimization. In *Scientific and Statistical Database Management, 1999. Eleventh International Conference on*, pages 264–273, Aug 1999.

[29] D. W. R. Cyganiak and M. L. (Ed.). RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, February 2014. http://www.w3.org/TR/rdf11-concepts/.

[30] M. Schmachtenberg, C. Bizer, and H. Paulheim. Adoption of the linked data best practices in different topical domains. In *The Semantic Web–ISWC 2014*, pages 245–260. Springer, 2014.

[31] A. Schultz, A. Matteini, R. Isele, C. Bizer, and C. Becker. LDIF - Linked Data Integration Framework. In *COLD*, 2011.

[32] O. Udrea, D. R. Recupero, and V. Subrahmanian. Annotated rdf. *ACM Transactions on Computational Logic (TOCL)*, 11(2):10, 2010.

[33] M. Wylot, P. Cudre-Mauroux, and P. Groth. Tripleprov: Efficient processing of lineage queries in a native rdf store. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 455–466, Republic and Canton of Geneva, Switzerland, 2014. International World Wide Web Conferences Steering Committee.

[34] M. Wylot, J. Pont, M. Wisniewski, and P. Cudré-Mauroux. dipLODocus[RDF]: short and long-tail RDF analytics for massive webs of data. In *Proceedings of the 10th international conference on The semantic web - Volume Part I*, ISWC'11, pages 778–793, Berlin, Heidelberg, 2011. Springer-Verlag.

[35] A. Zimmermann, N. Lopes, A. Polleres, and U. Straccia. A general framework for representing, reasoning and querying with annotated semantic web data. *Web Semant.*, 11:72–95, Mar. 2012.