Network-Aware Workload Scheduling for Scalable Linked Data Stream Processing

Lorenz Fischer, Thomas Scharrenbach, Abraham Bernstein

University of Zurich, Switzerland*
{lfischer,scharrenbach,bernstein}@ifi.uzh.ch

1 Introduction

In order to cope with the ever-increasing data volume, distributed stream processing systems have been proposed. To ensure scalability most distributed systems partition the data and distribute the workload among multiple machines. This approach does, however, raise the question how the data and the workload should be partitioned and distributed. A uniform scheduling strategy—a uniform distribution of computation load among available machines—typically used by stream processing systems, disregards network-load as one of the major bottlenecks for throughput resulting in an immense load in terms of inter-machine communication.

We propose a graph-partitioning based approach for workload scheduling within stream processing systems. We implemented a distributed triple-stream processing engine on top of the Storm realtime computation framework and evaluate its communication behavior using two real-world datasets. We show that the application of graph partitioning algorithms can decrease inter-machine communication substantially (by 40% to 99%) whilst maintaining an even workload distribution, even using very limited data statistics. We also find that processing RDF data as single triples at a time rather than graph fragments (containing multiple triples), may decrease throughput indicating the usefulness of semantics.

2 Problem Statement and Definitions

A linked data stream processing system essentially continuously ingests large volumes of temporally annotated RDF triples and emits the results again as data stream. Such systems usually implement a version of the SPARQL algebra that has been modified for processing dynamic data. The processing model considered is a directed graph, where the nodes are algebra operators and data is sent along the edges. Hence, each query can be transformed to a query tree of algebra expressions – the topology of the processing graph.

* The research leading to these results has received funding from the Europ. Union 7th Framework Programme FP7/2007-2011 under grant agreement no 296126 and from the Dept. of the Navy under Grant NICOP N62909-11-1-7065 issued by Office of Naval Research Global.

In order to scale the system horizontally (i.e., executing its parts on multiple processing units concurrently) we may replicate parts (or the whole) of the query's topology and execute clones of the operators, i.e., *tasks* in parallel. The workload of the system, can then be distributed across several machines in a compute cluster. We refer to the assignment of tasks to machines as *scheduling*.

The goal of our approach is to find a schedule (i.e., assignment of tasks to machines) for a given topology that minimizes the total number of data messages transferred over the network, whilst maintaining an even workload distribution across machines in terms of CPU cycles.

Many stream processing platforms attempt to uniformly distribute compute loads possibly incurring high network traffic. Approaches like Borealis [1] schedule the processors according to the structure of the query, where every operator is is assigned to one machine. This approach has an upper limit in parallelization equal to the number of operators and may incur high network traffic between two machines containing active operators. As Aniello et al. in their recent work [2], we propose to partition the data, to parallelize the operators, and then to minimize network traffic allowing for more flexibility for distributing the workload. In contrast to previous work, we don't try to implement a new scheduling algorithm *but much rather make use of existing graph partitioning algorithms to optimize the amount of data sent between machines.*

Hypothesis: *Combining data partitioning between tasks with a scheduler that employs graph partitioning to assign the resulting task instances outperforms a uniform distribution of data and tasks to machines.*

Our hypothesis assumes that different distribution strategies significantly influence the number of messages sent between the machines.

3 Evaluation and Results

We evaluated our system using two example queries that are built around a real world streaming use case: *SRBench* [5], (streams of weather measurements), and an open government dataset (self-acquired from public sources), which combines data on public spending in the US with stock ticker data.[1] We devised a query that would highlight (publicly traded) companies, that double their stock price within 20 days and are/were awarded a government contract in the same timeframe. The data for both use cases has been converted to and consumed as time annoated n-triple records.

Procedure First, we partitioned each dataset and compiled the queries into execution topologies. We then recorded the number of messages that were sent between tasks at runtime. Second, to test our hypothesis, we needed to partition the resulting communication graph based on the network load of each channel. Since the channel loads are not known before running the query we chose two experimental scenarios. In the first scenario we assume an *oracle* optimizer that would know the number of messages that would flow along every channel. This scenario allows to establish a hypothetical upper bound of quality that our

[1] http://www.usaspending.gov, https://wrds-web.wharton.upenn.edu/wrds

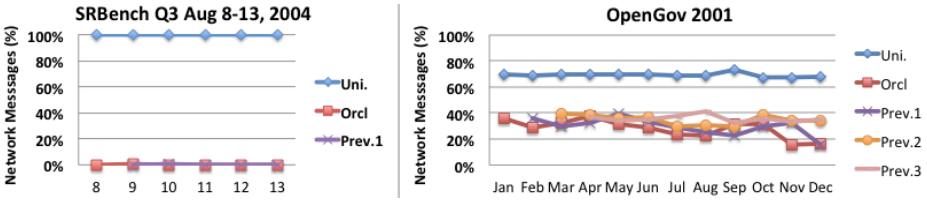

Fig. 1. Percentage of messages sent over the network for the uniform distribution and the graph partitioned setup, using either the test data itself (oracle) or data from the previous one to three time-slices as input for the graph partitioning algorithm.

method could attain, if it were to have an oracle. In a second scenario we assumed a *learning* optimizer that first observes channel statistics for a period of time and then partitions the graph accordingly. To that end we sliced the *SRBench* data into daily and the *OpenGov* data into monthly slices. We then measured the the performance of our approach based on learning during the preceding one to three time-slice essentially providing a adaptively learning system.
Third, to partition the graph we employed *METIS* [3]. We used the *gpmetis* in its standard configuration, which creates partitions of equal size, and only changed the *-objtype* parameter to instruct *METIS* to optimize for total communication volume when partitioning, rather than minimizing on total edgecut.

The Suitability of Graph Partitioning for Scheduling *The critical element for optimizing the scheduling using graph partitioning is that the operators can be parallelized with an adequate data partitioning.* The results show that using a graph partitioning algorithm to schedule task instances on machines does indeed reduce the number of messages sent over the network (Fig 1). We graph the number of network messages divided by the number of total messages as a measure for the optimality of the distribution. The *SRBench* data can be optimally partitioned by the id of the reporting weather station even when using only the data of the immediately preceeding time slice (left side, Prev.1). Once this task has been achieved, which we got due to the pre-partitioned datasets, all computation can be managed on a local machine, as no further joins are necessary. This clearly indicates that some queries can be trivially distributed when a good data partition is either known or can be learned.

For the *OpenGov* dataset (right side) the tested join operation requires a significant redistribution of messages. First, we find that our approach clearly outperforms the uniform distribution strategy by a factor of two to three. Second, even longer learning periods, using two (Prev.2) and even three previous time slices (Prev.3), do not necessarily improve the overall performance - maybe due to over-fitting or concept drift [4]. We also found that this only leads to a slightly less even load distribution.

For the *SRBench* query we observed a reduction in network usage by over 99%. For the *OpenGov* query, workload distribution using a graph partitioning approach yields savings in terms of network bandwidth of over 40%.

Balancing Computation Load In order to make good use of the available resources, a distributed system should assign equal workloads to all machines.

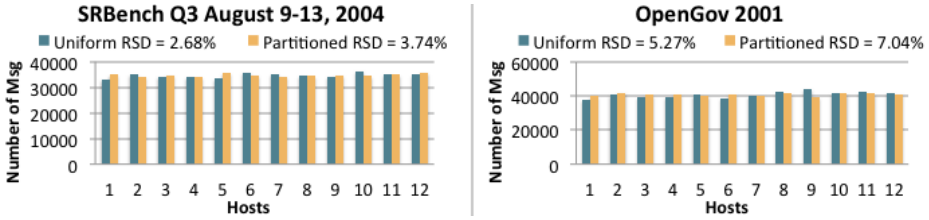

Fig. 2. Average computation load distribution for all time-slices of each dataset. RSD = Relative Standard Deviation

For this reason we analyzed how many messages were processed by all tasks of each partition for the two queries (Figure 2). The load distribution resulting from the graph partitioned task assignment only differs slightly from the one found by uniform task distribution (average relative standard deviation (RSD) *OpenGov*: 7.04% for partitioning vs. 5.27% for uniform baseline; SRBench: 3.74% for partitioning vs. 2.68% for uniform baseline).

The most important shortcomings of our study are its limitation to two datasets and queries and the fixed setup of the distributed system. For the first we intend to systematically extend our evaluation in the future in terms of number of datasets and queries. For the latter, is it the interactions between number of machines and cores available and the degree of parallelism that require further research. Especially the impact of such interactions on throughput in terms of messages ingested per second is of interest here. Future work will also investigate whether the principle of finding the smallest possible data partition given the desired degree of parallelism is as important as our experiments indicate.

We are confident that our findings help making DSFP systems more scalable and ultimately enable reactive systems that are capable of processing billions of triples or graph fragments per second with a negligible delay. It is our firm belief that the key to addressing these challenges needs to and will have to be revealed from the data itself.

Acknowledgements We would like to thank Thomas Hunziker, who wrote the first prototype of the *KATTS* system during his master's thesis in our group.

References

1. Abadi, D.J., Ahmad, Y., Balazinska, M., Hwang, J.h., Lindner, W., Maskey, A.S., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y., Zdonik, S.: The Design of the Borealis Stream Processing Engine. In: Proc. CIDR2005. pp. 277–289 (2005),
2. Aniello, L., Baldoni, R., Querzoni, L.: Adaptive online scheduling in storm. In: DEBS2013 (2013),
3. Karypis, G., Kumar, V.: A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM J. on Scientific Comp. 20(1), 359–392 (Jan 1998),
4. Vorburger, P., Bernstein, A.: Entropy-based Concept Shift Detection. In: Proc. ICDM2006. pp. 1113–1118 (2006)
5. Zhang, Y., Duc, P.M., Corcho, O., Calbimonte, J.p.: SRBench : A Streaming RDF / SPARQL Benchmark. In: Proc. ISWC 2012 (2012)