Synchronization Modeling in Stream Processing*

Marcin Gorawski and Aleksander Chrószcz

Abstract. Currently used latency models in stream databases are based on the average values analysis that results from Little's law. The other models apply theory of M/G/1 queuing system. Theses solutions are fast and easy to implement but they omit the impact of streams synchronization. In this paper, we introduce a heuristic method which measures the synchronization impact. Then we have used this solution to extend the popular model based on average values analysis. This modification allows us to achieve better accuracy of latency estimation. Because schedulers and stream operator optimization require a fast and accurate model, we find our model a good starting point to create better optimizers.

1 Introduction

A well-written and scheduled Data Stream Management System(DSMS) should meet deadlines and provide predictable performance. Real stream databases are highly complex systems which implement different optimization algorithms.

The latency and memory optimization has been widely analyzed in scheduler algorithms [8, 14, 13, 10, 7, 5, 2]. Also, big latency of tuples can result from temporary overload. At such moments, calculations cannot be processed fluently and stream queues rapidly become longer. Eventually, this situation may lead to a

Marcin Gorawski · Aleksander Chrószcz
Silesian University of Technology, Institute of Computer Science,
Akademicka 16, 44-100 Gliwice Poland
e-mail: {Marcin.Gorawski,Aleksander.Chroszcz}@polsl.pl

Marcin Gorawski
Wroclaw University of Technology, Institute of Computer Science,
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
e-mail: Marcin.Gorawski@pwr.wroc.pl

* Supported by EFS Grant POKL.08.02.01-24-019/08.

breakdown because of the shortage of storage resources. If a system specification allows a generation of incomplete results, we can randomly delete some tuples from data streams. Thanks to this, the amount of data to be processed is reduced and latency requirements can be satisfied for part of the data. The above solution is named load shedding and its usability was presented in [15, 3, 16].

The estimation of the average latency time is useful to tune schedulers which switch between time and memory optimization. This value is also helpful to adjust starting time when partial results should be generated when some data streams are temporarily not available. In the paper, we show that the synchronization of inputs in the presence of binary operators such as joins can cause a skew in the way tuples are introduced into downstream operators. This causes a substantial error of latency estimation which we want to reduce. We have run extensive experiments to find a model which can be easily applied in DSMS. Finally, we arrived at some heuristic model which describes the relations between utilization level, query definition and latency. Our model does not take into account moments when a node is temporarily overloaded beyond its CPU capacity. Our solution also does not focus on the impact of network links either. Despite those assumptions, its accuracy is bigger in comparison with popular average values analysis in DSMS.

The rest of this paper is organized as follows: Section 2 introduces the background of modeling stream queries; Section 3 describes the basic approach to modeling; Section 4 defines our theory of modeling stream queries; Section 5 explains how latency is calculated; next in section 6 our model is tested against competitive solutions; and finally Section 7 concludes the paper.

2 Basic Terms

Stream query Q is defined as a directed acyclic graph (DAG), whose nodes and arcs represent stream operators and streams respectively. Query Q is divided into sub queries $Q = \{u_1, u_2, ..., u_n\}$ and each of them is deployed on a calculation node in a distributed system. The calculation node is a separate processor or computer in a stream database. Besides, each stream operator has the following parameters defined:

1. Selectivity s_x is an average number of output tuples resulting from processing one input tuple by operator x.
2. Productivity u_x is the probability of generating at least one tuple by operator x when one input tuple is processed. Productivity and selectivity are equal for *selection*, *projection* and *map* operators. In contrast to them, a *join* operator's productivity and selectivity are different.
3. Processing cost c_x is the time required for an input tuple to be processed by operator x.

A single portion of data transmitted by a stream is a tuple. It has a timestamp which defines the time of its entry into a system. Each stream contains tuples ordered chronologically. Besides, each stream is described by the average tuple latency l. The time when a tuple enters a stream database and the time when this tuple causes the generation of a new tuple at the output of a given operator A constitute the beginning and the end of measured latency for operator A. In the paper, the latency is decomposed on three elements: operator processing time, synchronization time and queuing time.

We can classify stream operators according to the number of their input streams. Operators with one input stream are called unary operators, while operators with two input streams are labeled as binary operators. Stream operators process tuples in chronological order. Consequently, at each point of time instance an operator processes an input tuple with the smallest timestamp. When an operator is of binary type then the tuple with the smallest timestamp can be identified only when neither of the input streams is empty [11, 4].

In a stream database the time associated with an operator corresponds to a tuple timestamp which has been recently popped from its input stream. This time is named operator local time. From this viewpoint, the time flow is frozen between successive tuples. The shorter the interval between consecutive timestamps, the better the freshness of the local time associated with an operator and a stream. Let us define interval τ which is the time between those local time updates. There frequently exist a few tuples with the same timestamps in a stream. This is caused by operators which generate a few result tuples after processing one input tuple. As a consequence, if we want to measure interval τ between local time updates, we cannot divide the time of a stream observation by the number of popped tuples during this time. In the analysis of binary operators more complicated situation can be encountered. They process input tuples as long as both input streams are not empty. When one of them becomes empty, the processing is suspended. As a result, the operator time updates are divided into slots when stream processing is available. As a consequence, the distribution of output tuples can be described as groups of tuples separated by intervals. The bigger the interval between those groups is, the greater the number of tuples appearing per group. The illustration of such an effect is shown in fig. 1. We define the time of tuple group as a timestamp of the last tuple in the group. This metric informs us how often the time associated with a given operator and stream is updated. This is another way of representing data stream freshness [12].

Let us define global selectivity s_{path}. A sequence of operators which connects source a with operator b is defined as $path = \{a, ..., b\}$. The path which connects operators O_0 with O_3 in fig. 2 is defined as $\{O_0, O_2, O_3\}$. Then the global selectivity defined on $path$ equals $s_{path} = \prod_{i \in path} s_i$. Summing up, the value s_{path} represents the average number of tuples generated at the output of operator b when one tuple is processed through this $path$.

Besides, we also define set Src, which consists of source operators $1, ..., K$. The average output rate of those operators is defined by vector $X = (X^{(1)}, X^{(2)}, ..., X^{(K)})$.

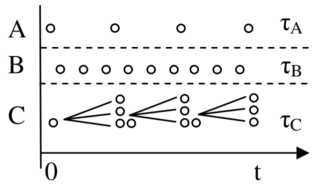

Fig. 1 The formation of groups of output tuples

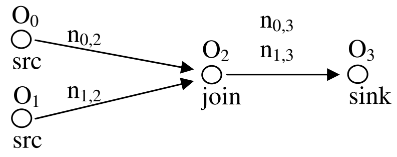

Fig. 2 Sample query graph

3 Basic Approach

The average values analysis that results from Little's law is the basic way of modeling queuing systems. This approach is popular in stream databases [9] because it is created only upon average metrics of query parameters. Our model assumes that a stream database consists of multiple processing nodes which serve multiple classes of clients; each processing node i evaluates sub query u_i.

Figure 2 illustrates a sample query which will be used to explain an operational analysis. Let us assume that $n_{a,b}$ is an average number of tuples flowing into operator b as a result of processing one tuple from source a. Besides, we define average tuple latency $l_{a,b}$ at the output of operator b which is a consequence of processing tuples from source a. Now, we want to estimate cumulative latency l_b, which represents the latency of all output tuples of operator b. Let us notice that the tuple latency of operator O_2 depends on which operator path a tuple has been processed on. For example in fig. 2, there exist two paths connected with operator O_2:$\{O_0, O_2\}$ and $\{O_1, O_2\}$. In order to find cumulative latency l_{O_2}, we have to estimate latencies for both paths and combine them. The formula below defines cumulative latency for operator b:

$$l_b = \frac{\sum_{a\in Src} X^{(a)} n_{a,b} l_{a,b}}{\sum_{a\in Src} X^{(a)} n_{a,b}} \tag{1}$$

Now we will estimate $l_{a,b}$. Each operator is directly or indirectly supplied by sources. The frequencies at which tuples from sources are transfered to operator o are defined by vector $X_o = (X_o^{(1)}, X_o^{(2)}, ..., X_o^{(K)})$. The utilization of calculation node i caused by operator o which processes tuples from source k equals:

$$U_i^{(o,k)} = X_o^{(k)} \bar{B}_i^{(o)} = X^{(k)} \bar{D}_i^{(o,k)} \tag{2}$$

where: $\bar{D}_i^{(o,k)} = B_i^{(o)} n_{k,o}$; $\bar{B}_i^{(o)}$ - an average time of processing a tuple by operator o deployed on processing node i.

The utilization of processing node i equals:

$$U_i = \sum_{o\in u_i} \sum_{k=1}^{K} U_i^{(o,k)} \tag{3}$$

This formula shows how long each operator located on processing node i calculates tuples which result from inserting tuples at source streams 1..K. The stream database is in a steady condition when all $U_i < 1$.

The average visit time of tuples queued to operator o at processing node i which results from processing one tuple from source k is:

$$\bar{R}_i^{(o,k)} = \frac{\bar{D}_i^{(o,k)}}{1 - U_i} \tag{4}$$

Finally, latency l_{path} is the sum of values by which latency increases when a tuple passes successive operators on the $path$. After processing one tuple from source k, a group of tuples can appear at operator o. We knew the average visit time at o which defines the time of processing this whole group of tuples. Now we want to find the relation between the average visit time and the value by which the latency is increased after passing operator o. In order to solve this we have assumed that tuples arrive evenly in time according to average throughput. Due to this arithmetic progression is applied to approximate the value by which the latency is increased at operator o: $\frac{\bar{R}_i^{(o,k)}}{2}$. Summing up, latency l_{path} is:

$$l_{path} = \sum_{o \in path} \frac{\bar{R}_{Node(o)}^{(o,k)}}{2} \tag{5}$$

where: $path$ - is the sequence of operators; $Node(o)$ - is the function which returns the processing node for stream operator o.

4 Estimation of Synchronization Impact

The analysis of synchronization impact is divided into two phases. At the beginning, we explain when stream operators require additional time to synchronize streams. The estimation of this time is based on the stream freshness property. In the next subsection, we introduce the algorithm used to calculate this value for each operator input stream.

4.1 Latency Caused by Synchronization

Figure 3 shows the notation of a binary operator and fig. 4 explains its parameters. The vertical markers in fig. 4 represent the moments when consecutive tuple groups arrive at inputs A and B. Variables τ_A and τ_B represent the intervals between those tuple groups for stream A and B respectively. The average latencies of tuples at the input of the analyzed operator are l_A and l_B. In other words, l_a is the amount of time before the tuple in stream A synchronizes with a tuple from stream B and l_b

is the time before the next tuple in stream B synchronizes with the corresponding tuple from stream A. It is worth noticing that we analyze only the sequence in which tuples are processed by a binary operator. We are not interested in the semantics of this binary operator. In order to make the description of the synchronization process easier to follow, τ_A and τ_B have similar values in fig. 4. As a result one tuple appears after another one arrives at the other input. Nevertheless, our model is not limited to this scenario.

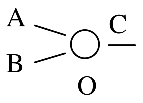

Fig. 3 The binary operator notation

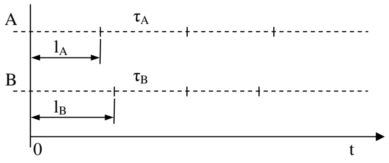

Fig. 4 Graphical representation of latency and τ

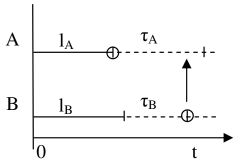

Fig. 5 Pessimistic scenario for input A

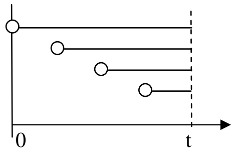

Fig. 6 The explanation of the latency calculation

If tuples arrive at input A with a latency greater than $l_B + \tau_B$ then stream B is not empty. Summing up, when $l_B + \tau_B < l_A$ then tuples at input A are processed directly. The other case is described by $l_B + \tau_B \geq l_A$ and illustrated in fig. 5. Tuples appear at input A with average latency l_A. Because the average interval between successive tuple groups equals τ_A, we assumed that one tuple appeared in the stream within τ_A with 100% probability. An analogous assumption is made for stream B.

The pessimistic scenario occurs when tuples arrive at input A at time l_A while tuples at input B arrive at time $l_B + \tau_B$. In this case tuple buffering lasts longest. Now we calculate the average latency for this pessimistic scenario. Let us notice that the sum of the latencies of all the tuples queued in stream A is the sum of the arithmetic sequence illustrated in fig. 6. Below we repeated the formula for the sum of elements in an arithmetic sequence: $S_n = a_1 + a_2 + ... + a_n = \frac{a_1+a_2}{2}n$. The average number of tuples which appear at input A between consecutive tuples at input B equals: $n_A = \frac{l_B+\tau_B-l_A}{\tau_A}$ In consequence, we arrive at the following formula: $l_{AC} = \frac{l_B+\tau_B-l_A+l_B+\tau_B-l_A-n_A\tau_A}{2}(n_A-1)$ When we combine the above formulae, we achieve the following average latency for tuples from input A.

$$l_{AC} = \frac{l_B - l_A + \tau_B}{2} \tag{6}$$

Summing up, the stream synchronization of binary operators introduces additional latency, which can be estimated by the following rule.

$$l_{AC} = \begin{cases} 0 & when \quad l_B + \tau_B < l_A \\ \frac{l_B - l_A + \tau_B}{2} & otherwise \end{cases} \tag{7}$$

4.2 Stream Freshness

The average interval τ informs us how frequently the time of a stream is updated. Knowing this, we can calculate the output latency of the operators attached to those streams.

Figure 1 explains why tuple groups are inserted into output streams. Let us assume the average interval between tuple groups in stream A is three times longer than the corresponding interval in stream B. On average, three tuples from input B are consumed at the time of the tuple arrival at input A.

Let us assume that input A started producing tuples and input B started generating tuples after x time units. When $\tau_A > \tau_B$ then we should consider each $x \in (0, \tau_B]$ so as to cover all possibilities. Because tuples are generated evenly, we have limited our observation to the time which passes from the appearance of one tuple from the slower stream to appearance of another one. During this time n tuples in the faster stream appear waiting $\tau_A - x$; and one tuple appears in the slower stream and it waits x time units. Now we can estimate the average interval between tuple groups for a given x and it equals the weighted average value of x and $\tau_A - x$.

$$\tau_C(x) = \frac{x + n(\tau_A - x)}{1 + n} \tag{8}$$

When we assume that x appears with equal probability in the range from 0 to τ_B, we arrive at the formula.

$$\tau_C = \frac{1}{\tau_0} \int_{x=0}^{\tau_0} \frac{x + n(\tau_1 - x)}{1 + n} dx = \frac{-2.5\tau_0^2 + \tau_0\tau_1 - (2\tau_0^2 + \tau_0\tau_1)\ln\frac{\tau_1}{\tau_0 + \tau_1}}{\tau_0} \tag{9}$$

where: $\tau_0 = \min(\tau_A, \tau_B); \tau_1 = \max(\tau_A, \tau_B)$.

This formula allows us to simulate the average interval between groups of tuples depending on the metrics of input streams. Then this interval is important in calculating a latency increase for consecutive operators.

The graph in fig. 7 shows the value τ_C for different ratios τ_A/τ_B. It is worth noticing that the average interval between consecutive groups of tuples is halved when the ratio value ranges from 0.6 to 1. When the ratio value ranges from 0 to 0.6, then the resulting average interval is closer to τ_B. Moreover, the operator is prone to generate peaks of overload.

The broken plot in fig. 7 illustrates a hypothetical case when the binary operator is free of the synchronization burden. A situation like this occurs when the timestamps of all tuples are known a priori. In such a case we divide the period of observation

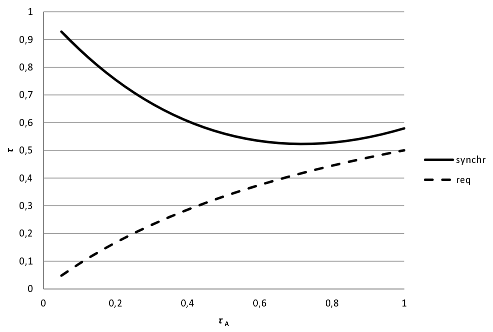

Fig. 7 The average interval between groups of tuples for changing proportion τ_A/τ_B

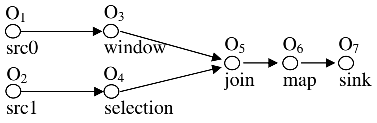

Fig. 8 Query DAG1

τ_B by the number of tuples which were processed by an operator. There is one tuple in stream B and $\frac{\tau_B}{\tau_A}$ tuples in stream A. Summing up, the average interval between groups of tuples is approximated as follows:

$$\tau_C = \frac{\tau_B}{1+\frac{\tau_B}{\tau_A}} \tag{10}$$

The comparison of both plots in fig. 7 shows that the synchronization of binary operators substantially affects the result latency. In order to process streams fluently it is important to have short intervals between groups of tuples. As it is shown in fig. 7, this cannot be achieved when the τ_A/τ_B ratio is under 0.6.

5 Average Latency

The mixed approach combines the impact of synchronization with the operational approach. The query DAG1 shown in fig. 8 will be used to explain the algorithm of latency estimation. Let us assume that operator o has attribute $visited$ which equals $false$ at the beginning of the calculation. There also exists function $next(o)$ which returns the set of operators connected to the output of operator o. Analogously we define function $prev(o)$, which returns the set of operators supplying tuples to operator o.

The latency estimation is a simple bottom-up calculation of $\bar{R}_i^{(o,k)}$ for each operator o and each source k as it is described in section 3. Having used this algorithm upon DAG1, we can achieve the following sequence of calculation: $O_1, O_2, O_3, O_4, O_5, O_6, O_7$. Next, we evaluate latencies on the paths connecting sources with each operator of the query according to alg. 1. Each operator o is described by the following properties: τ - it is an interval between consecutive groups of tuples; L - it is a map of latencies indexed by the source of stimulation.

At the beginning of alg. 1 method $initializeValues(o)$ setups values of source operators. This method gets the value of throughput $X^{(o.id)}$ for operator o, then assigns properties:

1. $o.\tau = \frac{1}{X^{(o.id)}}$
2. $o.L[X^{(o.id)}] = 0$

The consecutive steps of alg. 1 evaluate properties for unary and binary operators. Method $updateOp1(o)$ retrieves operator o_s which supplies operator o with tuples. Next the following properties are calculated:

1. For each source k: $o.L[k] = o_s.L[k] + \frac{\bar{R}_i^{(o.id,k)}}{2}$
2. $o.\tau = \frac{o_s.\tau}{o.u}$

The step 2 of the above method is necessary to model the extension of τ when an operator has low productivity. Method $updateOp2(o)$ retrieves operators: op_{s1} and op_{s2} which supply operator o with tuples. Next, the remaining steps of the method are processed:

1. Latency $l_{s1}(l_{s2})$ is calculated. The output rate for operator o is vector $Y = (Y_o^{(1)}, Y_o^{(2)}, ..., Y_o^{(K)})$. Those rates are divided according to data source k. Finally, l_{s1} is defined as:
$$l_{s1} = \frac{\sum_{k \in K} Y_{s1}^{(k)} o_{s1}.L[k]}{\sum_{k \in K} Y_{s1}^{(k)}} \tag{11}$$
Analogically l_{s2} is calculated.
2. Formula (7) is used to calculate values $l_{s1,o}$ and $l_{s2,o}$
3. The latency is calculated as follows: $l_o = \frac{(l_{s1}+l_{s1,o})a_{s1}+(l_{s2}+l_{s2,o})a_{s2}}{a_{s1}+a_{s2}}$ where $a_{s1}(a_{s2})$ represents the average number of tuples generated by operator $o_{s1}(o_{s2})$ during a time unit.
4. Value $o.\tau$ is updated according to formula (9).
5. For each source k: $o.L[k] = l_o + \frac{\bar{R}_i^{(o.id,k)}}{2}$

When the evaluation of alg. 1 is completed, the latency of a given operator can be calculated by means of formula (11).

Algorithm 1 (Latency evaluation)
```
foreach(op:A) {
  if(op is Source) {
    initializeValues(op);
  } else if(op is unary operator) {
    updateOp1(op);
  } else if(op is binary operator) {
    updateOp2(op);
  }
}
```

6 Tests

The experiments described in this section were conducted on our stream data base simulator which is the result of our previous experience with stream database StreamAPAS. Thanks to this, we were able to isolate the impact of dataset rates or other processes which share the same node. We prepared two types of datasets. One consists of tuples whose timestamps are distributed according to the Poisson process. The other set is based on the time distribution measured in real system [1].The sizes of those datasets were between 100'000 and 800'000 tuples. We calculated average metrics for each dataset and use it during empiric and analytic calculations. Each query has been run multiple times with changing level of utilization, as a result we verified the accuracy of our model for a range of configurations.

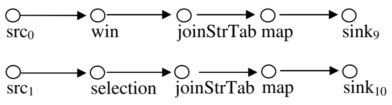

Fig. 9 Query DAG2

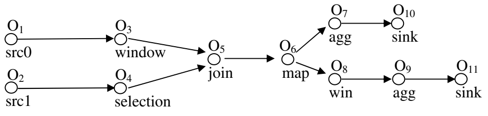

Fig. 10 Query DAG3

Let us now define the notation used in the following figures. A graph name with the suffix 'basic' labels graph which depicts estimation based on the operational approach. A graph name with the suffix 'mixed' shows estimation according to our mixed approach. The remaining graphs with the suffix 'simulation' indicate empirical results. Our aim was to create queries with different topologies. Because of the page limits we confront the empirical results with the analytic estimations for queries DAG1-DAG3, which are shown in figures: 8, 9 and 10. In fig.12 we can see that our mixed model substantially outperforms the popular basic approach that is used by current schedulers and optimizers. In order to measure the impact of binary operator synchronization. We have defined query DAG2 which replaces binary operator O_2 with unary operators with the same selectivity. Figure 13 shows that

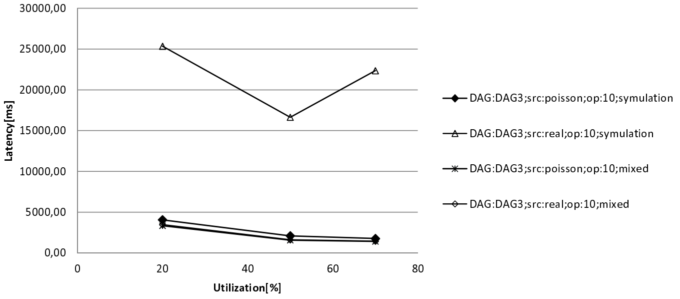

Fig. 11 Comparison of DAG3 results for different datasets

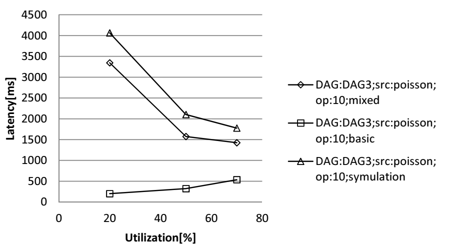

Fig. 12 Comparison of DAG3 results for the operational approach and the mixed approach

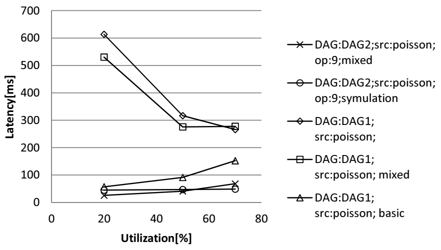

Fig. 13 Comparison of the synchronization impact between DAG2 and DAG1

the latency will be reduced from 300ms to values below 80ms if only synchronization could be avoided. Finally, we have tested how latency changes when real time distribution of data stream is applied. This effect is depicted in fig. 11.

The main conclusion drawn from our analysis of collected results is that the accuracy of our model is substantially higher than that of the basic approach. Despite the fact that our model is based on the analysis of average values, it offers nearly ideal precision for datasets generated by the Poisson process. The estimation accuracy is lower for datasets based on the real distribution of timestamps. This is caused by the fact that we measured average values for wide time windows. If we shorten those time windows then the assumption that data streams are generated by Poisson process will generate smaller error and we can achieve better estimation accuracy.

7 Conclusions

Multi-criteria optimization is a challenge in stream databases. When we consider a distributed system, then we have to monitor the utilization of the processing nodes. We can compose stream operators so as to optimize memory consumption but this optimization strategy affects the model of synchronization. As a consequence the memory optimization changes the latency of result tuples.

These above circumstances make the development of the analytical model important for future stream database systems. Currently the analysis based on average values is the most popular in such systems. Unfortunately, our tests show that this approach is weak when it comes to calculating the latency of result tuples. In the paper we have introduced an additional component, which reflects the impact of synchronization. The new metric in this component is the productivity property defined for each stream operator. By means of this we are able to more efficiently estimate the impact of synchronization. Thanks to this the exactness of latency estimation is substantially improved in comparison with the operational approach. What seems important, is that our component is based on average statistics, which makes our model easy to calculate in real-time systems. Our mixed model joins the analysis of utilization and latency in distributed stream databases which is a good foundation for the future schedulers. Besides, this model can be used to predict places where processing of a stream query can be easily destabilized.

During our further research we plan to create an optimizer which controls the frequency of boundary tuple [6] injection. In contrast to current algorithms which try to achieve the shortest latency, we want to achieve the predefined value. The next area of our interest is the creation of a scheduler which uses our model to strike a balance between memory and time optimization.

References

1. `http://ita.ee.lbl.gov/html/contrib/lbl-pkt.html`
2. Babcock, B., Babu, S., Datar, M., Motwani, R., Thomas, D.: Operator scheduling in data stream systems. The VLDB Journal 13(4), 333–353 (2004)
3. Babcock, B., Datar, M., Motwani, R.: Load shedding for aggregation queries over data streams. In: ICDE 2004: Proceedings of the 20th International Conference on Data Engineering, p. 350. IEEE Computer Society, Washington, DC (2004)
4. Bai, Y., Thakkar, H., Wang, H., Zaniolo, C.: Optimizing timestamp management in data stream management systems. IEEE 23rd International Conference on Data Engineering, ICDE 2007, pp. 1334–1338 (2007)
5. Bai, Y., Zaniolo, C.: Minimizing latency and memory in dsms: a unified approach to quasi-optimal scheduling. In: SSPS 2008: Proceedings of the 2nd International Workshop on Scalable Stream Processing System, pp. 58–67. ACM Press, New York (2008)
6. Balazinska, M.: Fault-tolerance and load management in a distributed stream processing system. Ph.D. thesis, Cambridge, MA, USA (2006); Adviser-Hari Balakrishnan
7. Carney, D., Çetintemel, U., Rasin, A., Zdonik, S., Cherniack, M., Stonebraker, M.: Operator scheduling in a data stream manager. In: VLDB 2003: Proceedings of the 29th International Conference on Very Large Data Bases, pp. 838–849. VLDB Endowment (2003)
8. Hammad, M.A., Franklin, M.J., Aref, W.G., Elmagarmid, A.K.: Scheduling for shared window joins over data streams. In: VLDB, pp. 297–308 (2003)
9. Hwang, J.H., Xing, Y., Çetintemel, U., Zdonik, S.B.: A cooperative, self-configuring high-availability solution for stream processing. In: ICDE, pp. 176–185 (2007)
10. Jiang, Q., Chakravarthy, S.: Scheduling Strategies for Processing Continuous Queries over Streams. In: Williams, H., MacKinnon, L.M. (eds.) BNCOD 2004. LNCS, vol. 3112, pp. 16–30. Springer, Heidelberg (2004)
11. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21(7), 558–565 (1978)
12. Sharaf, M.A.: Metrics and algorithms for processing multiple continuous queries. Ph.D. thesis, Pittsburgh, PA, USA (2007)
13. Sharaf, M.A., Chrysanthis, P.K., Labrinidis, A.: Preemptive rate-based operator scheduling in a data stream management system. In: AICCSA 2005: Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications, pp. 46–I. IEEE Computer Society, Washington, DC (2005)
14. Sharaf, M.A., Chrysanthis, P.K., Labrinidis, A., Pruhs, K.: Efficient scheduling of heterogeneous continuous queries. In: VLDB 2006: Proceedings of the 32nd International Conference on Very Large Data Bases, pp. 511–522. VLDB Endowment (2006)
15. Tatbul, E.N.: Load shedding techniques for data stream management systems. Brown University, Providence (2007); Adviser-Zdonik, Stan
16. Tatbul, N., Çetintemel, U., Zdonik, S., Cherniack, M., Stonebraker, M.: Load shedding in a data stream manager. In: VLDB 2003: Proceedings of the 29th International Conference on Very Large Data Bases, pp. 309–320. VLDB Endowment (2003)