Collaborative Streaming: Trust Requirements for Price Sharing

Tobias Grubenmann
Department of Computer Science
The University of Hong Kong
Hong Kong SAR, China
tobias@cs.hku.hk

Daniele Dell'Aglio
Department of Informatics
University of Zurich
Zurich, Switzerland
dellaglio@ifi.uzh.ch

Abraham Bernstein
Department of Informatics
University of Zurich
Zurich, Switzerland
bernstein@ifi.uzh.ch

***Abstract*—Stream Processing (SP) is an important Big Data technology enabling continuous querying of data streams. The stream setting offers the opportunity to exploit synergies and, theoretically, share the access and processing costs between multiple different collaborators. But what should be the monetary contribution of each consumer when they do not trust each other and have varying valuations of the differing outcomes? In this article, we present Collaborative Stream Processing (CSP), a model where the costs, which are set exogenously by providers, are shared between multiple consumers, the collaborators. For this, we identify three important requirements for CSP to establish trust between the collaborators and propose a CSP algorithm, ENCSPA, adhering to these requirements. Based on the collaborators' outcome valuations and the costs of the raw data streams, ENCSPA computes the payment for each collaborator. At the same time, ENCSPA ensures that no collaborator has an incentive to manipulate the system by providing misinformation about her/his value, budget, or time limit. We show that ENCSPA can calculate payments in a reasonable amount of time for up to one thousand collaborators.**

***Index Terms*—Trust, Big Data, Stream Processing, Cost Sharing**

I. Introduction

High *velocity* is one important dimension of Big Data [1]. Consequently, the field of Stream Processing (SP) [2]–[4] has been proposed to continuously analyze data as it arrives at the processing entities. The Internet of Things [5] and the Semantic Web [6] are two examples of platforms where streaming plays an important role. We can observe some significant differences between streaming data and static data: (1) streams are produced and consumed continuously, (2) the computation can be distributed over various loosely connected machines, and (3) queries are evaluated over long time periods and thus, part of the processing may overlap.

These differences between static and streaming data lead us to the idea of *Collaborative Stream Processing* (CSP), a model where different consumers, or *collaborators*, share the cost and outcome of Stream Processing (SP). The most naïve approach to share the cost of SP is to assign an equal cost share among all collaborators and give everyone access to all the processed data. Such a solution works as long as all collaborators are of the same type, that is, all collaborators have equal value for the outcome, equal budget, and equal lifetime for their queries. In practice, however, it is unlikely that different collaborators coincide in all these different dimensions. Consequently, we propose a cost sharing model where the financial contribution of each collaborator depends on (1) what kind of data is required to answer the streaming query, (2) how long the query is executed, (3) the maximum budget of each collaborator, and (4) the maximum value of each collaborator. The collaborators in our model are assumed to be self-interested, in particular, we do not assume that the different parties will *trust* each other. Therefore, the *CSP algorithm*—this is the algorithm which determines the price shares based on the collaborators' streaming queries and other required parameters—has to establish *trust* in the price sharing procedure. For this, we identify in Section IV-A three requirements which are paramount in establishing trust.

We introduce the problem through an example of processing Big Data with high velocity: the processing of smart city data. Municipalities might be interested in smart waste collection, civic associations in monitoring pollution, and navigation services in computing desirable paths considering real-time traffic information. Each of the three entities—municipalities, civic associations, and navigation services—have their own streaming query. However, there might be some overlap in the computation they perform on these data and hence, some output can be reused for several different queries. As a result, computational resources can be saved by doing the respective computation only once and, in turn, municipalities, civic associations, and navigation services can save some money by collaborating and thus, sharing the cost of the computation.

In this article, we study the question: *How much should each collaborator contribute to the payment for gathering and processing the various pieces of data that answer their differing but overlapping queries in the absence of trust?* As a solution, we propose CSP to study the cost sharing of continuously processing data streams. The main contributions are: (1) we introduce a formal model, Collaborative Stream Processing, for cost sharing in stream processing (which to our

This work and in particular Tobias Grubenmann were partially supported by the Swiss National Science Foundation under grant #153598. Tobias Grubenmann is also supported by the Innovation and Technology Commission of Hong Kong (ITF project MRP/029/18).

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

knowledge is the first attempt to do so), (2) we present three requirements that a CSP algorithm must fulfil to build trust into the system, (3) we propose, ENCSPA, a CSP algorithm that maximizes the data consumer's benefits (or utility), (4) we study how ENCSPA fulfills the aforementioned requirements, and (5) simulate the runtime and the savings of money to empirically show that ENCSPA is able to calculate payments in a reasonable amount of time for up to one thousand collaborators.

II. Related Work

Stream Processing: As the name suggests, Stream Processing (SP) engines [3] process streaming data. SP solutions can broadly be grouped in Data Stream Management Systems (DSMS) [2] and Complex Event Processing (CEP) systems [4]. The former ones typically introduce sliding windows to convert portions of the streams into relations, which can be managed with classical relational algebra operators, whilst the latter ones analyze the stream to detect relevant event patterns (e.g., sequences of desired events).

The Web, and in particular the Semantic Web, offers another distributed and decentralized environment where steaming data is produced, processed and consumed. The Semantic Web community developed extensions to manage data streams and continuous queries [6]. These new solutions enabled new and interesting applications. In [7], Wagner et al. describe a scenario where information from energy producers, grid operators, and appliance manufacturers must be collected and processed. Smart city projects [8], [9] aim at processing stream data produced by the daily city life to optimize the city's operational tasks.

Financial Cost Sharing: Shapley defined in [10] two key axioms for cost sharing: (1) the cost shares should be additive dependent on the total costs and (2) if a participant demand does not increase the overall cost, the cost share should be zero. In [11], Aumann and Shapley introduced the Aumann-Shapley rule, which charges each participant prices based on the integral of the marginal cost. This pricing rule was successfully applied in [12] to a telephone billing problem. Serial Cost Sharing [13] is a pricing method which is robust to coalition deviations if participants are not allowed to redistribute their outputs. More recently, in [14], a method is introduced to divide up the costs of a network when different participants require different subsets of the resources. Our method expands the equal need cost share defined in [14].

None of the related work addressed cost sharing in a setting where different participants require the resources for different, overlapping time periods and participants can potentially manipulate the model by misinformation.

Cloud Computing and Publish/Subscribe Models: A different approach to reducing the cost of stream processing are cloud-based solutions. In [15], Ishii et al. show that costs of stream processing can be reduced up to 80% while taking into account the trade-off between the application's latency and the costs of using the cloud infrastructure. The distributed stream processing platform ESC [16] is able to dynamically request and release machines at runtime and hence, reducing the cost of stream processing by buying only as much cloud computing resources as needed.

While cloud computing solutions also focus on reducing costs by sharing resources, their approach is orthogonal to the cost sharing solution proposed in this paper. Indeed, our approach can be deployed on the cloud.

Publish/subscribe models [17] target loosely coupled interactions between distributed systems on the internet scale. In such models, subscribers have the chance to subscribe to specific event or event patterns and are subsequently notified from publishers matching their request.

In [18], Jurca et al. proposed a system which allows users to subscribe to certain sensor attributes in a publish/subscribe manner. They studied how to decrease the overall network traffic of processing multi-join correlation queries by removing redundant subscriptions. In [19], XFlow is proposed, a system for distributed stream processing, which allows a user to minimize the global system cost. The strength of XFlow is that sources and destinations of data streams are decoupled using the publish/subscribe paradigm. While XFlow encourages the sharing of results of certain stream operators like in our model, the question of how to share the cost for these operations and avoid manipulation is orthogonal to the system implemented in XFlow. Indeed, Collaborative Stream Processing, presented in this paper, can be easily integrated into existing systems like XFlow.

None of the works mentioned above discuss the issue of trust between different entities sharing the financial cost of collaborating.

III. Motivating Example

Consider a setting where different sources for streaming data provide viewership data for TV channels, the Electronic Program Guide (EPG) data for those channels, and activity data from social networks. There are various potential collaborators who are interested in these data sources: TV channels can learn which TV shows are the most popular and improve their programming, advertisement networks can improve their advertising strategies based on the people's current interests, and providers of recommender services can improve their recommendations.

Since all these consumers are interested in the same underlying sources, there are opportunities for cost sharing. Figure 1 shows a possible scenario, where a TV channel, an ad network, and a recommender system are interested in the same data but require different computation on these data. The TV channel is only interested in the combination of viewership data and the EPG data, which results in data about the viewers of a specific show. The streaming query of the TV channel accesses this viewer-per-show data and processes it further. The ad network and the recommender system are interested in combining the viewer-per-show data with some data from social networks to include the social media responses. Each of the two, the ad network and the recommender system, does some additional computation on the data specific for their

individual requirement. As obvious from the figure, the costs of some computational nodes and some of the sources could be shared by the collaborators.

IV. Collaborative Stream Processing

We build our model, Collaborative Stream Processing (CSP), based upon the general architecture of stream processing [3], [20]–[22], which means that CSP can be applied to any instance of this general architecture, irrespective of the exact details of the implementation. In this general architecture, *sources* produce streams of data which are processed by operators producing new streams, which are eventually propagated to interested actors. A consumer can submit a query, which captures what computation should be performed. As the operators are processing data continuously over time, one can associate a cost per time for each operator. Likewise, a cost per time can be associated to each source that streams data into our CSP model. The cost per time for sources and operators is determined by the provider of the respective service.

Collaborative Stream Processing is a model for collaboration of different consumers—the collaborators—who decide to share the costs for their stream processing queries. In CSP, we do not make any assumption of how this collaboration is carried out. One of the collaborators could decide to collect and run all queries and charge the other collaborators, afterwards. Or, a cloud provider could establish a collaboration platform where different collaborators can organize themselves and execute the query in the cloud. Even whole platforms solely dedicated to matching collaborators with overlapping needs and providing them a platform to run their streaming queries are conceivable. In the following, we introduce the formal model of CSP and its notation. Table I lists the symbols used throughout this paper.

A *collaborator* is a consumer of streaming data who decides to share the cost of stream processing with other collaborators. We assume that we have $n \in \mathbb{N}$ collaborators, each having exactly one query. We denote with $Q = \{q_1, \ldots, q_n\}$ the set of all queries. As each collaborator has exactly one query, we identify a specific collaborator by the respective query $q_i \in Q$.

For each stream processing query $q_i \in Q$, the *value coefficient* $v_i \in \mathbb{R}_+$ indicates how much the collaborator is maximally willing to pay for each time unit (e.g., an hour, a day, or a month). The CSP platform has to determine the unit of the value coefficient. For the remainder of this paper, we assume that the associated unit of the value coefficient is *dollars-per-day* [$/d]. For example, a value coefficient of $\$5/d$ means that the collaborator is maximally willing to pay $5 for running the query for one day, $10 for running the query for two days, and $2.5 for running the query for half a day.

In addition to the value coefficient, we also need to know the *budget* of each collaborator. For each query $q_i \in Q$, the budget $b_i \in \mathbb{R}_+ \cup \{+\infty\}$ indicates how much a collaborator having query q_i is maximally willing to pay for running the query, irrespective of the runtime. We assume that the unit of the budget is *dollars* [$]. If a collaborator does not have any constraints on the budget, then b_i equals $+\infty$. For example, a budget of $10 means that a collaborator is not willing to pay more than $10, irrespective of whether the query runs for a day, a second, or a month.

Finally, the *time limit* $T_i \in \mathbb{R}_+ \cup \{+\infty\}$ indicates how long the collaborator having query $q_i \in Q_i$ is interested in processing the streaming query, at most. The time limit acts as an upper bound on the runtime of the query. We assume that the unit of the time limit is *days* [d]. Like the budget, the time limit can take the value $+\infty$, indicating that the collaborator does not want to restrict the runtime of the query. For example, a collaborator might want to restrict the runtime of a query to a week, because he/she has no interest in the streaming data after this period.

The combination of value coefficient, budget, and time limit allows a collaborator control over which allocations of runtime and payments would be beneficial for her/him and which allocations would not. Given this information, CSP defines for each query $q_i \in Q_i$ an *allocated runtime* $t_i \in \mathbb{R}_+$ and an *allocated payment* $\pi_i \in \mathbb{R}_+$. The allocated runtime determines how long the specific query will be executed. For the remainder of this paper, we assume that the unit of the allocated runtime is *days* [d]. The allocated payment determines the amount of money the collaborator has to pay. We assume that the unit of the allocated payment is *dollars-per-day* [$/d]. The effective payment from each collaborator is the allocated payment times the allocated runtime (fractions of the time unit, e.g. 0.1 days, are allowed). In CSP, each collaborator having query $q_i \in Q_i$ has a (quasi-linear) *utility* $u_i \in \mathbb{R}$, which is the value for the allocated runtime for each query minus the payment she/he has to pay:

$$u_i := v_i \cdot t_i - \pi_{i,j} \ .$$

Each query $q_i \in Q$ requires some sources and operators to create the desired output. The CSP derives a query-plan for each query $q_i \in Q$. We denote with $O_i = \{o_{i,1} \ldots, o_{i,j_i}\}$ the $j_i \in \mathbb{N}$ nodes which are required for query $q_i \in Q$. A *node* $o_{i,j} \in O_i$ can be either a source or an operator. Note that we do not distinguish between sources and operators in CSP, as only the cost associated with either of them is of relevance. We denote with $\gamma(o_{i,j})$ the *cost* of node $o_{i,j} \in O_{i,j}$. The cost of a node is the total amount of money required to keep the node operational. This includes the cost of the computational resources as well as the associated network costs. We assume that the unit of the costs is *dollars-per-day* [$/d]. Furthermore, we will denote with $O = \bigcup_{i=1}^{n} O_i$ the set of all nodes in the CSP.

A *CSP algorithm* is an algorithm which calculates the allocated runtime t_i and allocated payment π_i, given the nodes O_i, value coefficient v_i, time limit T_i, and budget b_i for each query q_i. In the following sections, we first define the requirements we want a CSP algorithm to fulfil, followed by ENCSPA, a CSP algorithm which calculates allocated runtimes and payments for a given set of queries and parameters.

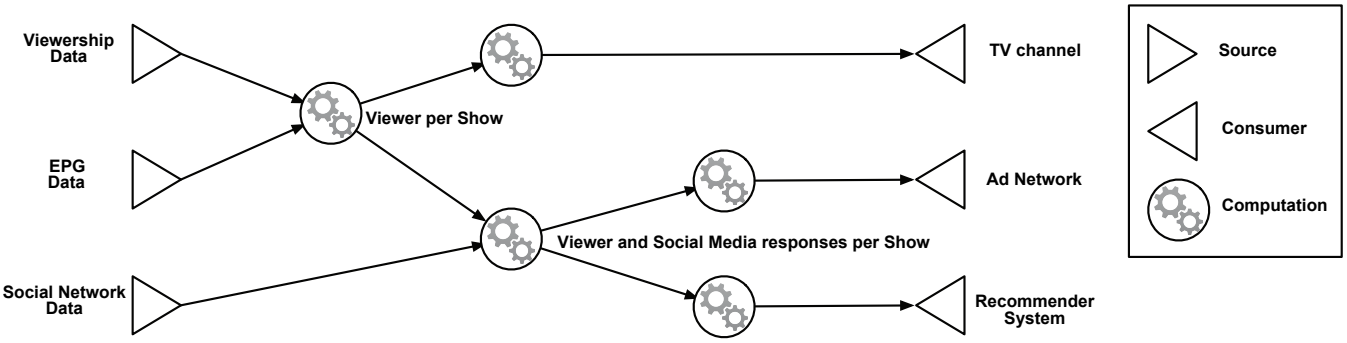

Fig. 1. TV channels, ad networks, and recommender systems are interested in the same data but require different computation. The arrows indicate the flow of data.

TABLE I
SYMBOLS USED IN THIS PAPER

Symbol	Meaning
$Q = \{q_1, \ldots, q_n\}$	The set of n queries.
$v_i \in \mathbb{R}_+$	The value coefficient for q_i.
$b_i \in \mathbb{R}_+ \cup \{+\infty\}$	The budget for q_i.
$T_i \in \mathbb{R}_+ \cup \{+\infty\}$	The time limit for q_i.
$t_i \in \mathbb{R}_+$	The allocated runtime for q_i.
$\pi_i \in \mathbb{R}_+$	The allocated payment for q_i.
$u_i \in \mathbb{R}$	The utility of the collaborator having query q_i.
$o_{i,j} \in O_i$	A node of q_i.
$\gamma(o_{i,j}) \in \mathbb{R}_+$	The cost of node $o_{i,j}$.
$\gamma_{q_i\|Q'}(o_{i,j}) \in \mathbb{R}_+$	The cost share of node $o_{i,j}$ for q_i given $Q' \subseteq Q$.

Afterwards, we discuss the properties of ENCSPA, including to which degree it meets our requirements.

A. Requirements for Trust in Collaborative Stream Processing

We identify three requirements that a CSP algorithm should satisfy in order to establish trust from the collaborators.

Respecting collaborators' constraints: First, the CSP algorithm has to respect the constraints set by all the collaborators. This is essential for any CSP algorithm, as a collaborator cannot directly influence the allocated runtime and payment and hence, needs to rely on the algorithm to allocate only within the boundaries specified by the collaborator. These constraints are: *value coefficient*, *budgets*, and *time limits*:

[R1] The CSP algorithm must respect each collaborator's value coefficient, budget, and time limits.

Benefits from participating: It is key to ensure that each collaborator benefits from its participation in CSP. Each collaborator should have as least as high utility as she/he would have by running her/his query in isolation, without collaborating and sharing costs with others. In other words, the benefit should consist of an increment in utility, whenever this is possible due to overlapping sources or operators. This leads to our second requirement:

[R2] A collaborator must have at least as high utility as running the query outside of CSP and strictly higher utility when there is some overlap of sources or computation between her/his query and the other ones.

This requirement is crucial to ensure that collaborators gain advantage for participating in CSP. Without this guarantee, a collaborator might not receive a fair price share and thus, might overpay, which would harm the trust in CSP. Note that Requirement R2 assumes that the collaborator has strictly higher utility than running the query outside of the CSP, if there is any potential for cost sharing. In the case that a query does not have overlapping computation or sources with other queries, there is no opportunity for cost sharing and the collaborator should have at least as high utility as when not participating in CSP.

No benefits from misreporting parameters: The CSP algorithm needs the following parameters as input from each collaborator: *value coefficient*, the *budget*, and the *time limit*. As these parameters influence the collaborator's utility, we have to investigate how collaborators could benefit by misreporting these parameters. Misreporting in this context means reporting a parameter which deviates from the collaborator's true parameter in order to gain an advantage. Requirement R3 makes the CSP algorithm robust against manipulations from the collaborator with respect to value coefficient, execution time limit, and budget:

[R3] No collaborator must be able to benefit by misreporting value coefficient, time limit, or budget.

If the CSP algorithm would encourage such manipulations, directly or indirectly, it would enable some collaborators to increase their utility by gaming the model. The problem is that such gains in utility are always at the expense of fellow

collaborators' utilities. The collaborators need to have trust in the CSP algorithm that no collaborator can gain an unfair advantage by misinformation.

B. ENCSPA, a CSP Algorithm

The Equal-Need-Collaborative-Stream-Processing-Algorithm (ENCSPA) is a CSP algorithm which uses equal-need cost sharing [14]. ENCSPA determines (1) the *allocated runtime*, i.e., how long each query should be executed, and (2) the *allocated payment*, i.e., how much the collaborator should pay.

First, we must discuss how the cost of a single node can be shared among different collaborators. To distribute the cost of a single node, we use *equal-need cost sharing* [14]. The idea of equal-need cost sharing is: everyone requiring a specific resource should contribute equally to the cost of this resource. Applying this to our setting, the cost $\gamma(o)$ for each node $o \in O$ should be equally distributed among all queries requiring the node. Let $\gamma_{q_i|Q'}(o)$ be the *cost share* for node $o \in O$, which is the amount that the collaborator having query $q_i \in Q'$ has to contribute towards the cost of node $o \in O$ when sharing the costs with other queries $Q' \subseteq Q$. Then:

$$\gamma_{q_i|Q'}(o) = \frac{\mathbb{1}_{o \in O_i}}{\sum\limits_{O_j \in O | q_j \in Q'} \mathbb{1}_{o \in O_j}} \cdot \gamma(o) \ ,$$

where $\mathbb{1}_{\text{condition}}$ is the indicator function which equals to 1 if the condition is true and to 0, otherwise.

Furthermore, we define

$$\gamma_{q_i|Q'}(q_i) := \sum_{o \in O_i} \gamma_{q_i|Q'}(o)$$

as the cost share of query q_i.

Note that as a direct consequence of *equal-need* cost sharing is that if one collaborator only requires a subset of the data requested by another collaborator, the former pays a reduced price, as the latter requires more nodes to compute the result.

Example 1: Figure 2 shows an example of how the costs are shared. The collaborator having query q_3 should contribute to financing the cost of the bottom two sources and the bottom two computation nodes. The costs of the two sources and the first computation node is shared with query q_2, the cost of the last computation node is covered by q_3 alone. Consequently, the payment for running q_3 is $\pi_3 = 0.5 \cdot \$3/\text{d} + 0.5 \cdot \$1/\text{d} + 0.5 \cdot \$1/\text{d} + \$2/\text{d} = \$4.5/\text{d}$. Note that this payment π_3 might change if time limits and budgets are introduced.

Algorithm 1 shows how the allocated runtime and the allocated payment are computed given the value coefficients, budgets, and time limits of the collaborators.

Algorithm 1: ENCSPA

Data: Set of queries Q. Set of nodes O. Value coefficients v_i, budgets b_i, and time limits T_i, for each $q_i \in Q$

Result: Allocated runtime t_i and allocated payments π_i for each query $q_i \in Q$.

```
1  t_1, ..., t_n ← 0
2  π_1, ..., π_n ← 0
3  S ← Q
4  while S ≠ ∅ do
5      do
6          for q_i ∈ S do
7              p_i ← γ_{q_i|S}(q_i)
8              if v_i < p_i then
9                  S ← S \ {q_i}
10     while A query has been removed from S
11     τ_min ← min_{q_i ∈ S} ( min( b_i / p_i , T_i ) )
12     for q_i ∈ S do
13         t_i ← t_i + τ_min
14         T_i ← T_i − τ_min
15         π_i ← b_i + τ_min · p_i
16         b_i ← b_i − τ_min · p_i
17         if b_i ≤ 0 or T_i ≤ 0 then
18             S ← S \ {q_i}
19 return t_1, ..., t_n, π_1, ..., π_n
```

C. Properties of ENCSPA

The time complexity of ENCSPA is $O(n^2)$, where n is the number of collaborators. To see this, one has to observe that each iteration of the outer while loop (Line 4–18) and the inner while loop (Line 5–10) removes at least one collaborator in each iteration from S. Therefore, each for-loop (Lines 6–9 and Lines 12–16) runs at most n times, and each for-loop has itself a time complexity of $O(n)$. The space complexity is $O(n)$ as the algorithm must keep track of a constant number of values for each collaborator.

In the remainder of the section, we study to which degree ENCSPA fulfils the requirements.

Requirement R1: ENCSPA meets the first requirement: The cost share p_i is always smaller or equal than the corresponding value coefficient v_i (Line 8) and any increment in allocated runtime τ_{min} is bounded by the budget and time limit of each collaborator (Line 11).

Requirement R2: To see that each consumer benefits from participation (R2), note that ENCSPA guarantees that a query shares costs only from those operators and sources relevant for the query. Hence, the assigned cost share can never be higher than the cost of running the query in isolation, in which case the collaborator of the isolated query has to cover all the costs alone. In addition, if there is any overlap in common sources or operators with other queries, the assigned cost share will be strictly cheaper than running the query in isolation.

Next, it is important to note that ENCSPA allocates runtime and charges payments as long as the assigned cost share is smaller or equal than the consumer's value coefficient. This means that the consumer's utility is maximized, given the information provided and the assigned cost shares, because as long as the value coefficient is still higher or equal than the

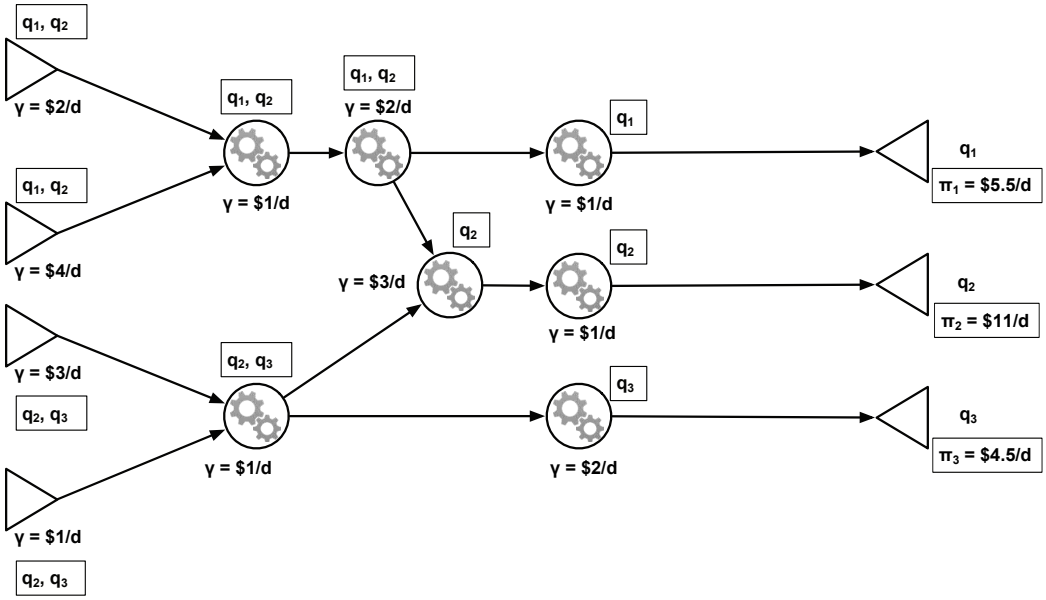

Fig. 2. Sharing of costs among different queries. The framed labels indicate all the queries for which the source/operator is relevant and the associated cost for each query.

costs, any additional runtime increases the overall utility by the difference between the value coefficient and the cost share. In addition, ENCSPA never allocates a runtime to a query which would exceed the runtime limit or would cause to surpass the budget. Since the collaborator's utility is maximized given the assigned cost shares, and the assigned cost shares are smaller (or equal if there is no sharing possible), the collaborator's utility must be smaller (or equal) than running the query in isolation. Hence, each collaborator profits from ENCSPA.

Requirement R3: A CSP algorithm must make sure that a collaborator cannot gain utility by misreporting about her/his *value coefficient*, *budget*, or *time limit*. As ENCSPA guarantees to maximize the collaborators' utilities given the assigned cost shares, a collaborator can only profit from misreporting if this yields a lower allocated payment. While ENCSPA takes into account the parameters given by the collaborator, these parameters are only used to determine if, and for how long a given query is executed. Using equal need cost sharing guarantees that the cost shares themselves are independent of these parameters. Hence, the allocated payments cannot be influenced by misreporting and, consequently, no collaborator can benefit from this.

V. Simulation

The aim of the simulation is to study the runtime behavior and financial advantages of ENCSPA. The runtime of ENCSPA is a crucial metric because the cost shares need to be updated whenever a new query arrives in the system. The total number of participating queries is limited by how fast ENCSPA can integrate new queries by recalculating the cost shares. The more queries are participating in CSP, the more money each participant can potentially save.

1) Data: We created randomly generated graphs, as described below. We used a fixed model for the continuous queries where each query consists of seven operator nodes and eight source nodes. Each operator has two inputs and one output. The computational nodes are arranged in three layers. In total, there are 500 sources available in our scenario. We create a number of n different queries. Each newly added query has for each operator node an *overlap probability* p that a particular node coincides with an existing node belonging to the same layer. If two nodes coincide, their child nodes in subsequent layers do as well. In addition, we created 500 source nodes from which each operator node in the lowest layer randomly chooses 2. The number of sources is fixed and does not grow with the number of queries in the evaluation.

2) Code: For the purpose of this evaluation, the algorithm was implemented in Java and executed on a PC with i5-8250U processor and 16GB of RAM. The code for the evaluation and the generation of the random graphs is online available at https://doi.org/10.5281/zenodo.3529213. Figure 3 shows the time needed to do the calculation of the cost sharing and the allocated runtime.

3) Experiments: We focused on the budget as the constraining parameter. The results for other parameters are comparable.

Each operator node and each source has a cost of \$1/d. The value coefficient and the time limit are set to infinite for each query. This means that the budget is the constraining parameter. The budget for each query is a uniformly randomly assigned value between 0 and 1, which prevents different queries having the exact same allocated runtime, a situation that would be very unrealistic and would improve the runtime of the allocation algorithm.

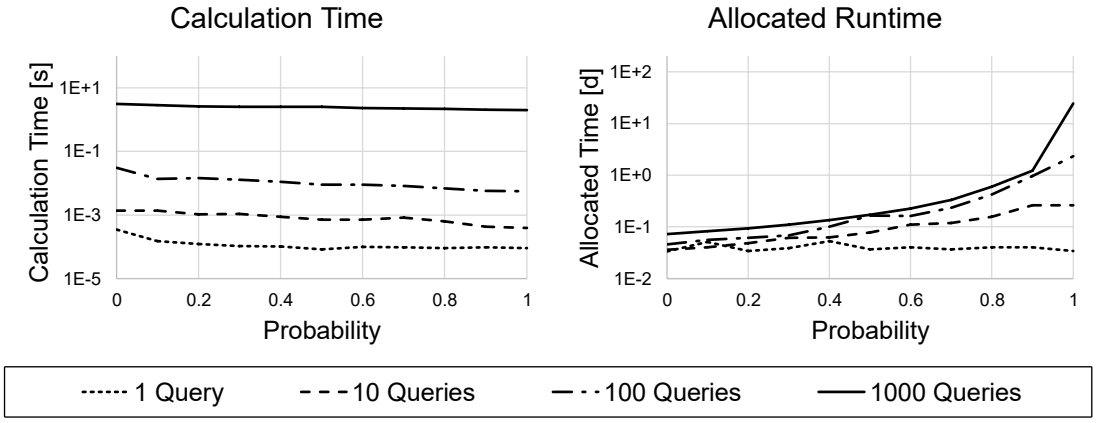

Fig. 3. Calculation time and allocated runtime.

The left graph in Figure 3 shows how the calculation time varies for different probabilities and over different number of queries. As we can see, the calculation time does not vary much when varying the overlap probability p but does vary much when varying the number of queries. The right graph shows how much runtime a query gets allocated on average for different overlap probabilities and different number of queries. Note that a query gets more allocated runtime the more overlap the queries have and the more queries are present in our model. The reason for this is that more overlap clearly means more sharing between queries and hence, more money savings for each query. The higher the number of queries, the higher is the positive influence of overlapping computation, as there are more queries which share the costs.

VI. Limitations and Conclusion

The growing interest in streaming data creates new opportunities to save money by sharing costs. We studied scenarios where different continuous queries share the costs, which are exogenously given by the providers of common operations and common sources. As we have seen, consumers can profit from Collaborative Stream Processing. Our proposed CSP algorithm, ENCSPA, allows one to calculate payments which ensure participation and discourage manipulation. To establish trust in CSP, we identified three requirements. We also showed that ENCSPA respects these three requirements and thus, shows that it is feasible to implement them in a practical setting. At the best of our knowledge, ENCSPA is the first algorithm that adheres to these requirements. As such, our contribution paves the way for research on establishing realistic cost sharing approaches for stream processing settings.

References

[1] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International Journal of Information Management*, vol. 35, pp. 137–144, Apr. 2015.

[2] B. Babock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings Sysmposium on Principles of Database Systems*, 2002.

[3] G. Cugola and A. Margara, "Processing flows of information: From data stream to complex event processing," *ACM Computing Surveys*, vol. 44, no. 3, pp. 15:1–15:62, 2012.

[4] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston, MA: Addison-Wesley, 2001.

[5] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[6] D. Dell'Aglio, E. Della Valle, F. van Harmelen, and A. Bernstein, "Stream reasoning: A survey and outlook," *Data Science*, vol. 1, no. 1–2, pp. 59–83, 2017.

[7] A. Wagner, S. Speiser, and A. Harth, "Semantic web technologies for a smart energy grid: Requirements and challenges," in *Proceedings of the International Semantic Web Conference ISWC '10*, pp. 33–37, 2010.

[8] D. Puiu, P. Barnaghi, R. Tönjes, D. Kümper, M. I. Ali, A. Mileo, J. X. Parreira, M. Fischer, S. Kolozali, N. Farajidavar, F. Gao, T. Iggena, T.-L. Pham, C.-S. Nechifor, D. Puschmann, and J. Fernandes, "Citypulse: Large scale data analytics framework for smart cities," in *IEEE Access*, vol. 4, pp. 1086–1108, 2016.

[9] S. Tallevi-Diotallevi, S. Kotoulas, L. Foschini, F. Lécué, and A. Corradi, "Real-time urban monitoring in Dublin using semantic and stream technologies," in *Proceedings of the International Semantic Web Conference ISWC '13*, pp. 178–194, 2013.

[10] L. S. Shapley, "A value for n-person games," in *Contributions to the Theory of Games II*, vol. 28 of *Annals of Mathematical Studies*, Princeton University Press, 1953.

[11] R. J. Aumann and L. S. Shapley, *Values of Non-Atomic Games*. Princeton, New Jersey: Princeton University Press, 1974.

[12] L. J. Billera, D. C. Heath, and J. Raanan, "Internal telephone billing rates–a novel application of non-atomic game theory," *Operations Research*, vol. 26, no. 6, pp. 956–965, 1978.

[13] H. Moulin and S. Shenker, "Serial cost sharing," *Econometrica*, vol. 60, no. 5, pp. 1009–1037, 1992.

[14] H. Moulin and F. Laigret, "Equal-need sharing of a network under connectivity constraints," *Games and Economic Behavior*, vol. 72, pp. 314–320, 2011.

[15] A. Ishii and T. Suzumura, "Elastic stream computing with clouds," in *2011 IEEE 4th International Conference on Cloud Computing*, pp. 195–202, July 2011.

[16] B. Satzger, W. Hummer, P. Leitner, and S. Dustdar, "Esc: Towards an Elastic Stream Computing Platform for the Cloud," in *2011 IEEE 4th International Conference on Cloud Computing*, (Washington, DC, USA), pp. 348–355, IEEE, July 2011.

[17] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, pp. 114–131, June 2003.

[18] O. Jurca, S. Michel, A. Herrmann, and K. Aberer, "Processing publish/subscribe queries over distributed data streams," in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems - DEBS '09*, (Nashville, Tennessee), p. 4, ACM Press, 2009.

[19] O. Papaemmanouil, U. Çetintemel, and J. Jannotti, "Supporting Generic Cost Models for Wide-Area Stream Processing," in *2009 IEEE 25th International Conference on Data Engineering*, (Shanghai, China), pp. 1084–1095, IEEE, Mar. 2009.

[20] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, "Monitoring streams: a new class of data management applications," in *VLDB '02 Proceedings of the 28th international conference on Very Large Data Bases*, pp. 215–226, 2002.

[21] A. Margara, J. Urbani, and F. van Harmelen, "Streaming the web: Reasoning over dynamic data," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 25, 2014.

[22] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM SIGMOD Record*, vol. 34, no. 4, pp. 42–47, 2005.