Abstract

Data stream processing systems (DSPSs) compute real-time queries over constantly changing streams of data. A stream is a possibly infinite sequence of tuples, or timestamped data items. In contrast to traditional databases, in which queries are issued over stored data, results in DSPSs are generated continuously as new data enters the system.

The constant increase in data volume renders the provisioning of a DSPS difficult and costly. It may require computing resources that may not be available or too costly to purchase. Even if a user opts for a cloud deployment, thus renting all resources on demand, perfect processing may still not be possible due to the financial cost.

In the future we can expect the development of processing infrastructures in which different parties cooperate to the creation of large-scale federated resource pools. The correct allocation of resources on these federated pools is going to be difficult, as the management of the single clusters will be be under the control of the local authority, often leading to unbalanced resource allocations and the local occurrence of overload. For these reasons, *overload* should be considered a common operating condition for such DSPS and not an exception.

Overload can be considered a type of failure because the system is not able to fully carry out the required computation. A system operating under constant overload is thus subject to continuous failure. In this situation, the system needs to discard some of its input data, an operation called *load shedding.*

Many streaming applications are able to produce valuable results even after some failure has occurred during the processing. Examples of such applications are meso-scale weather prediction, tornadoes and hurricanes forecasting, and real-time social media monitoring. An approximated result may still be useful to the user, as long as it is delivered with a low latency and it contains some information about its quality. In many cases, an imperfect result is better than no result at all.

We propose a new stream processing model under overload. The system constantly estimates the impact of overload on the computation and reports to the user the achieved quality-of-service. We introduce a quality metric called *Source Information Content (SIC).* This can be used by the user as an indicator for the achieved quality-of-service and by the system to implement intelligent shedding policies and to better allocate the system resources among users. When an overloaded system performs *load-shedding*, the choice of how much and what to discard is crucial for the correct functioning of the system. The SIC quality metric helps the system make more informed decisions when to shed data. It allows the implementation of a *fair-shedding* policy, giving an equal quality-of-service to all users, without penalising certain kinds of queries.

We develop these ideas as part of a research prototype called DISSP, the Dependable Internet-Scale Stream Processing engine. With it we explore the issues related to overload management and fair resource allocation. We show that augmenting streams with the SIC metric allows the system to make better load-shedding decisions, leading to more accurate results for many queries. It also allows the user to reason about the amount of processing resources that are needed to run a given query, striking a balance between the quality of the delivered results and the cost of operating the system.

Chapter 1

Introduction

Data stream processing systems (DSPSs) compute real-time queries over constantly changing streams of data [ScZ05]. A stream is a possibly infinite sequence of tuples, or timestamped entities [ABW06]. Differently than traditional databases, where queries are issued over stored data, in DSPSs queries are set first and results are generated as new data enters the system. This allows the generation of real-time updated results based on the constantly changing available data streams. Real-time in this context refers to the ability of delivering results *as they become available*. Stream queries are *continuous* [BW01], they process tuples as they are pushed into the system, delivering results as they are computed.

Internet-scale stream processing (ISSP) systems represent the next step in the evolution of stream processing [Pie08]. Similarly to how search engines make static web data useful to users, an ISSP

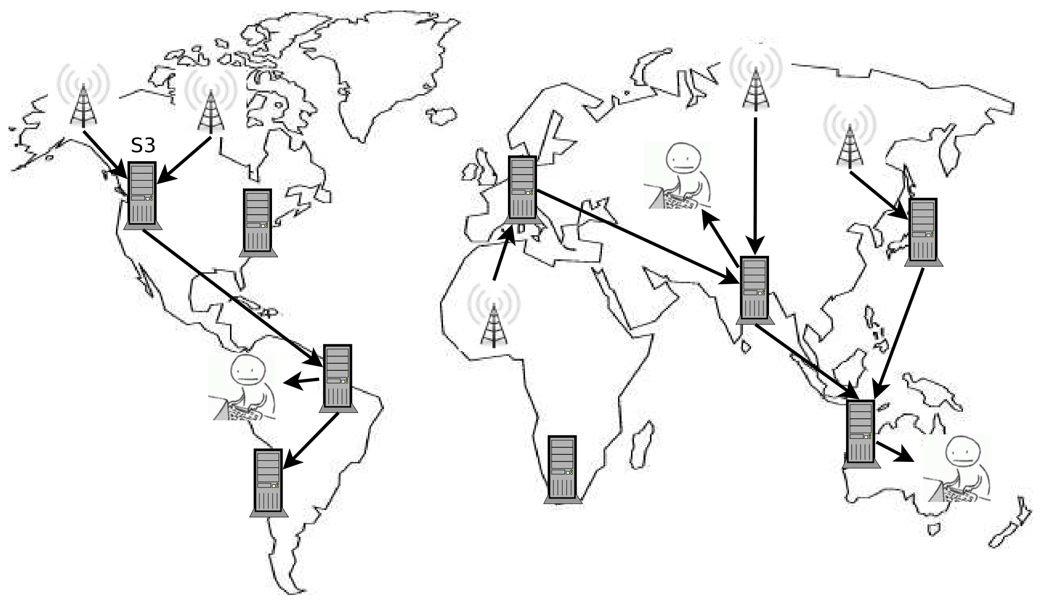

Figure 1.1: Graphical representation of an Internet-scale stream processing system at work.

system should reliably collect, filter and process stream data from potentially thousands of data sources on behalf of many users. Such systems should support a large number of queries, possibly running for extended periods of time. Data should be processed in a decentralised fashion, distributing the computation over several data centres.

In this context the notion of *sensor* is not only limited to a device measuring some physical quantities, but refers to a more extended range of devices. *Physical sensors* are devices taking measurements of some physical quantity at interval time, for instance a temperature sensor deployed in a sensor network, on the other hand a *software sensor* takes measurements about non-physical events, like social media events.

Figure 1.1 shows an ISSP at work. Sources, either physical or software, are located at different locations around the world (i.e. weather towers). Data streams are depicted as arrows. Several data centres, also located at different locations, are used to process these streams, in a decentralised fashion. Many users can concurrently issue queries, which the system tries to satisfy at the best of its possibilities, according to the available resources.

We can expect the rise of globally distributed ISSP systems in the future, in which several parties contribute processing resources to be shared by all users. These federated resource pools are going to join several clusters distributed across the globe, each administered by a local authority. Users will be able to deploy queries processing data coming from all over the world in real-time. Some systems started developing in this direction, like MaxStream [Bot+09], a federated stream processing infrastructure designed to support real-time business intelligence applications, and Cosm [CSM12], a system aiming at the construction a world-wide network of intelligent devices to build the Internet-of-Things.

1.1 Applications

These shared platforms can support the concurrent execution of a large number of queries, submitted by multiple users. The amount of available input data renders the provisioning of the processing resources difficult as it might be too costly to purchase or rent the necessary infrastructure. The variety of queries that a stream processing system can support is very broad and allows the support of a large number of applications. Among these, two interesting application domains that can be used as examples, deal with queries analysing *sensor data* and *social media* streams.

Sensor data. The advances in cheap micro-sensing technology is increasing the availability and resolution of sensor data. In the future the number of sensing devices will increase to a point where everything of material significance will be "sensor tagged" and will report its state and position in real-time. One area that has recently flourished in this field is environmental monitoring. Understanding how and why the climate is changing so rapidly is one of the top priority of many scientists and will be crucial to plan future policies. Sensor networks can also be employed to monitor in real-time weather phenomena that are potentially catastrophic, like hurricanes and tornadoes. The creation of large-scale weather sensing infrastructures will increase the possibility of identifying these phenomena before they become disruptive, thus allowing for an early response that will lead to a mitigation of their impact.

Social media. Social media represent a great source of user generated data, which offers new interesting mining possibilities. These data streams are populated with status updates, check-ins, photos, videos, etc., an ever increasing flux of everyday experiences that the users decide to share. The amount of available data is theoretically enormous and opens the doors to a new class of streaming applications. Dealing with the totality of the generated information might be over costly, while the analysis of a sample, capturing a certain percentage of the original stream, might be sufficient. Accepting the discarding of a portion of the input data allows a substantial reduction of the processing resources needed to run the queries and thus a reduction in the operational costs. What is important is that this reduction in quality of the generated results is quantified and reported to the user. This allows to establish a trade-off between the processing costs and the quality of services achieved by the query results.

In these application domains a result may be acceptable even if imperfect, as long as there is an indication about the entity of the quality degradation. The judgement about the quality of the delivered results should be left to the end user, while the system should provide constant feedback about the achieved quality of service.

1.2 Problem Statement

Sometimes the over provisioning of a stream processing system is difficult to achieve, like in the case of a federated resource pool, or might be too costly, like in a cloud deployment. When the amount of resources needed for perfect processing is not available, it is often possible to deliver meaningful results even processing only a subset of the input data. In many cases an imperfect result is better than no result at all.

The constant increase in data availability allows the computation of a richer variety of stream processing queries, but also renders the provisioning of the system resources difficult and costly. An extremely large amount of input data requires an amount of processing resources that might not be in the availability of the user or too costly to purchase. The cost of purchasing and administering the needed processing infrastructure can be too high and outside the financial possibilities of a user. In order to obtain a the needed processing capacity without having to own the entire infrastructure, it is possible to opt either for a *federated resource pool* or for a *cloud deployment*.

In a federated resource pool, many parties contribute their resources to create a larger shared infrastructure that can be used by all according to their needs. In this scenario each institution is only responsible for the costs of purchasing and administering their processing units, while being able to use the complete set of resources when available. These kinds of deployments, where many parties pool together their resources are subject to a phenomenon similar to the *tragedy of the commons* [Har68], since every party tends to consume more resources than what it contributed, the amount of available resources is always scarce. Another problem with such a deployment is that, when resources are scarce, it is only possible to increase the provisioning of the local domain, while the other processing sites are under the control of third parties.

Another way to acquire a large amount of processing resources is to rent them on a per need basis. It is possible in fact to opt for a cloud deployment, renting the required processing nodes from a cloud provider only for the time needed. In this way the cost of purchasing and maintaining the infrastructure is outsourced and the user pays only the resources it needs for a limited amount of time. This allows the acquisition of a large processing capacity for a reasonable cost and only for the time of the queries. The amount of resources needed, however, can be very large and the rental cost may be out of the financial reach of a user. Even if the user opted for a cloud deployment, renting all the resources needed, perfect processing might not be feasible due to financial constraints.

The loss of information that determines the quality reduction of results is due to either the loss of tuples or their intentional discarding. Processing nodes can fail and some might become temporarily unreachable because of a network failure, thus causing a loss of a portion of the processed tuples. In this scenario the stream processing system should be able to track the loss of information and to quantify its impact on the quality of the results. The user should be provided with a quality metric that reports in real-time the achieved quality of service, so that it can deliberate over the goodness of the provided output.

Another common reason that leads to the reduction of the processed information is the deliberated shedding of a portion tuples. This happens when the rate at which tuples are delivered in input to a processing node is greater than the rate at which it is able to process them. The throughput of the node reaches a plateau and can not increase any longer. Therefore, the excess tuples delivered in input need to be discarded, this process takes the name of *load-shedding*. In order to overcome this overload condition it is possible to increase the amount of processing resources, but, as explained earlier, this is might not be feasible or it could be preferable not to do so. Instead of trying to recover, it might be beneficial to accept the overload condition if the cost of running the processing is reduced while maintaining an acceptable quality of the results.

In a resource constraint deployment, with a large number of users and queries, the correct allocation of resources becomes critical. The problem is how to allocate resources fairly so that all queries achieve a similar quality of service, regardless of their nature. The system should be able to reason also about the relative performance of queries, so that, when it has to perform load-shedding, it can discard tuples in a fair fashion. Having a metric that captures the achieved quality of service for all queries, allows the system to implement a shedding policy that allocates resources fairly among queries. In this way the quality of the delivered results is equalised and queries receive the correct amount of processing resources according to their needs.

Is is possible to track the loss of information and report its entity to the user? Is there a correlation between the quality of the processing and the correctness of the delivered results? When the system have to choose what tuples to discard in an overload condition, is possible to design a fair resource allocation so that all queries experience the same quality of service? In a cloud deployment, where resources are rented on a per need basis, what is the trade-off between the cost of processing and the quality of the delivered results? These are the research questions that this thesis tries to address.

1.3 Quality-Aware Stream Processing

A stream processing system that operates under constant overload has to design with the ability to self-inspect and evaluate the quality of service it delivers to its users. When the amount of input data exceeds the processing capabilities of a node, the system has to discard a certain amount of tuples in input. By refusing to accept new tuples for a certain amount of time, the load on the system can be reduced and the the overload condition overcome. This approach, however, does not take into account the nature of the discarded tuples and results in a random shedding of input data.

When overload is not an exceptional condition, but is instead considered the normal operating condition, the system needs to be able to reason about the information carried by the individual tuples of each query. This allows the system to reason about what tuples it receives in input so that it can implement a semantic shedding policy. By augmenting tuples with a metadata value expressing their importance to the query calculation, it is possible to select tuples based on their information content and avoid resorting to a random shedding approach. This thesis proposes the introduction of a quality metric to quantify the amount of information captured by a tuple, tracking the degradation of information quality due to overload and failure.

1.4 Contributions

The choice about what tuples should be shed and what should be kept is crucial to the quality of the results. The system has to decide which tuples are more important for the processing and privilege these over others other tuples of lower quality. Not all tuples contain the same amount of information and thus they have a different value to the query. A tuple containing a value aggregated over a large number of sensors is probably more valuable than a a tuple containing a single reading. The system should be able to reason about the information content of tuples in order to identify the most valuable set of tuples to be spared from shedding. To do so it is necessary to introduce a *quality metric* that captures the quantity of information that went into the creation of a tuple and thus its value. Employing this metric the system is then able to implement a *semantic shedding policy*, an algorithm that allows to reason about the quality of tuples and to choose the most valuable to the computation.

SIC quality metadata. In order to allow the system to self-inspect and reason about the achieved quality of service, this work proposes the introduction of a quality metric called *Source Information Content (SIC)*. The purpose of this metric is to provide a hint about the amount of information contained in a tuple and thus to its importance for the current query results. It is added to streams in the form metadata, whose value is calculated by the query operators and is inversely proportional to the occurrence of failure. Employing the SIC metric the system is able to reason about the individual value of tuples and can implement intelligent resource allocation policies based on it. Chapter 3 introduces a quality-centric data model for stream processing system based on the SIC metric. It presents the assumptions behind the design of the SIC quality metric and explains how this is calculated by the system.

Semantic shedder. When overload is considered the normal operational mode of the system and not a rare, transient condition, it becomes crucial to choose intelligently what input tuples to discard in the load-shedding phase. If the rate at which tuples are received in constantly higher than the rate at which they can be processed, a node is in the condition of continuously having to discard a certain portion of its input tuples. Augmenting streams with the SIC quality metric provides a valuable indicator about the amount of information captured during the creation of a tuple and thus about its importance to the query processing. When performing load-shedding, the system can leverage this knowledge to implement a semantic shedding policy, an algorithm to choose the set of tuples to discard not at random, but according to specific criteria. Chapter 5 presents the design and implementation of a fair-shedding policy, which tries to allocate the system resources proportionally to the requirements of queries, so that the achieved quality of service is similar for all queries.

DISSP prototype design. The theoretical concepts introduced by the data model have been applied in the DISSP stream processing prototype. This system was designed to make use of the SIC quality metric at the core and is able to operate under heavy overload while tracking the quality of service achieved by every query. It also implements a semantic shedding policy that aims at providing a fair amount of resources to all queries. Overload should be considered a common operating condition on such systems and not an exception. Overload can be considered a kind of failure, since the system is not able to fully carry out the required computation. A system operating under constant overload is then subject to continuous failure. Instead of considering this failure as a rare, transient condition, the system should accept it and degrade its processing quality, while reporting to the user the achieved quality of service. In some cases it is better to reduce the quality of the delivered results in order to maintain the cost within the user's budget. Some classes of applications can still produce meaningful results even if some data is lost during the computation, since, in many cases, an imperfect result is better than no result at all.

Experimental evaluation. The first set of experiments investigated the correlation between the degradation of SIC values and the correctness of results. Even though the SIC quality metric was not designed as an accuracy metric, it was found that for many aggregate queries there is a good correlation between SIC value and correctness.

Having introduced the possibility of implementing semantic shedding policies, a second set of experiments was designed to test the advantages of employing the SIC quality metric in the load-shedding process. In particular the experiments focused in the comparison between a random shedder and the fair-shedding algorithm introduced in Chapter 5. It was found that employing the fair-shedding policy results in a better resource allocation among queries for many classes of queries.

Finally, a set of experiments tested the correlation between SIC degradation and reduction of cost. In this scenario the user voluntarily accepts a reduction of quality in the results, i.e. lower SIC values, in order to reduce the amount of required resources and thus the cost in a cloud deployment. The results showed that it is possible to strike a trade-off between a reduced SIC value of results and the costs of renting the processing infrastructure.

1.5 Document Outline

The rest of this document is organised as follows. Chapter 2 presents the background material relevant to this research. It describes some applications of stream processing, focusing on those that better tolerate approximated processing. It introduces different streaming data models and the basic concepts of stream processing. It presents different approaches to overcoming overload, focusing on the different load-shedding strategies available. It also look at techniques used to deal with failure, as consistency and replication.

Chapter 3 introduces a quality-centric data model, where streams are augmented with the SIC meta-data metric. This is a quality metric designed to capture the amount of failure that occurs in the generation of a tuple. It provides a hint about the amount of information captured by a tuple and thus about its contribution to the query results.

Chapter 4 presents the DISSP stream processing system, a research prototype that implements the proposed quality-centric data model. It explains how the abstract concepts introduced by the model has been implemented in a concrete system. It describes the design of the prototype and shows how it the SIC metric is calculated and used.

Chapter 5 focuses on the overload management process by analysing the load-shedding process. It introduces the concept of fair resource allocation among queries and a semantic shedding policy that implements it. It describes the algorithm used to discard tuples in an overloaded nodes so that all queries achieve a similar SIC value for their results.

Chapter 6 presents a set of experiments designed to evaluate some properties of the SIC quality metric. It investigates its correlation with the accuracy of results. It compares a random shedding policy with the proposed fair-shedding algorithm. Finally it looks at the trade-off between cost and SIC value degradation, offering the possibility to voluntarily accept imperfect results in order to reduce the operational costs.

Chapter 7 concludes the document and present a discussion on the research contribution of this work.