Revenue-based Resource Management on Shared Clouds for Heterogenous Bursty Data Streams

Rafael Tolosana-Calasanz[1], José Ángel Bañares[1], Congduc Pham[2], and Omer F. Rana[3]

[1] Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, Spain `rafaelt, banares@unizar.es`
[2] LIUPPA Laboratory
University of Pau, France `congduc.pham@univ-pau.fr`
[3] School of Computer Science & Informatics
Cardiff University, United Kingdom `o.f.rana@cs.cardiff.ac.uk`

Abstract. When data from multiple sources (sensors) are processed over a shared distributed computing infrastructure, it is necessary to often provide some Quality of Service (QoS) guarantees to each data stream. Service Level Agreements (SLAs) identify the cost that a user must pay to achieve the required QoS, and a penalty that must be paid to the user in case the QoS cannot be met. Assuming the maximisation of the revenue as the provider's objective, then it must decide which streams to accept for storage and analysis; and how many (computational / storage) resources to allocate to each stream in order to improve overall revenue. We propose an infrastructure for supporting QoS for concurrent data streams to be composed of self-regulating nodes. Each node features an envelope process to accept user streams; and a resource manager to enable resource allocation, admission control and selective SLA violations, while maximizing revenue.

Keywords: Data stream processing; Cloud computing; SLA Management; Admission control; QoS provisioning

1 Introduction

The number of applications that need to process data continuously over long periods of time has increased significantly over recent years. Often the raw data captured from the source is converted into complex events – which are subsequently further analysed. Such applications include weather forecasting and ocean observation from sensors [**?**], text analysis (especially with the growing requirement to analyse social media data, for instance), "Urgent Computing" [**?**], and more recently data analysis from electricity meters to support "Smart (Power) Grids" [**?**]. Data source (sensor) nodes can vary in complexity from smart phones to specialist instruments, and can consist of sensing, data processing and communication components. Data streams in such applications are generally large-scale and distributed, and generated continuously at a rate that cannot be estimated in advance. Scalability remains a major requirement for such applications, to handle variable event loads efficiently. Data elements are streamed from their source to their sink, and may be processed en-route (referred to *in transit* processing), rather than entirely at source /destination [?]. The benefit of such an approach is many fold: (i) to reduce power consumption at source (which may have limited battery capacity) and sink (which may have limited data storage space); (ii) enable the outcome of data analysis to be shared between multiple users; (iii) enable optimization by moving the processing close to the producers and consumers where applicable (consider an example where there are multiple sensors within the same location, and the event processing involves aggregation of events that are emitted by these sensors); (iv) alter the processing rate at intermediate (in transit) nodes to achieve a particular QoS requirement [?]; (v) combine data streams with archived data at intermediate nodes; (vi) enable fault tolerance to be supported at intermediate nodes – thereby providing an overall resilient infrastructure that masks faults generated due to the generation of large data volumes (referred to as data inflation) or failure of resources involved in data processing [?]; (vii) redirect event traffic to different nodes according to network conditions or workload [?].

We assume an Event Processing Network (EPN) is composed of a sequence of stages and datasets are streamed through that sequence (pipeline). Each EPN stage is mapped to a node in the infrastructure, though a node can enact more than one EPN stage. Using this approach, *each node* must be able to *self-regulate* its behaviour dynamically through adaptive resource provisioning, i.e. resources allocated for each incoming stream can be varied dynamically by a node controller. Various existing approaches [?,?,?,?] identify how Cloud infrastructures can be used to support data stream analysis. When multiple applications are executed within the same shared elastic infrastructure, each stream must be isolated from another and for the underlying coordination mechanism to adapt the infrastructure to either: (i) run all instances without violating their particular Quality of Service (QoS) constraints; or (ii) indicate that, given current resources, a particular instance cannot be accepted for execution. The QoS demand of each stream is captured in a Service Level Agreement (SLA) – which must be pre-agreed with each service provider prior to analysis. Such an SLA identifies the cost that a user must pay to achieve the required QoS and a penalty that must be paid to the user if the QoS cannot be met [?].

In previous works, we presented scenarios to validate the use of TB and the adaptation of TB parameters to achieve SLA objectives. In [?], a scenario that shows the role of TB for *shaping* the amount of data that subsequently forwarded to computational resources was presented, allowing for bursts of data while at the same time providing isolation of data streams. In [?], we subsequently demonstrate how a streaming pipeline, with a variation in the amount of data generated (referred to as data inflation/deflation), can be supported and managed using a dynamic control strategy at each node. Finally, in [?], we analyse revenue models for supporting streaming under the presence of faulty computational resources. These papers show main models developed using Petri nets to specify the EPN and architectural components and the engine to enact them[?]. In this paper, we extend previous work [?] by characterising a revenue model for in-transit analysis. We propose the use of the TB model for estimating the resources required to meet the SLA of each incoming data stream, extending our previous use of the TB mechanism for event traffic shaping. The remainder of this paper is structured as follows. Section **??** describes revenue models for in-transit analysis and the characterization of resource requirements for QoS in event processing applications. Section **??** shows the system architecture based on the token bucket model and a rule-based SLA management of QoS. Section **??** shows our evaluation scenario and simulation results. In Section **??**, the related work is briefly discussed. Finally, conclusions are given in Section **??**.

2 Revenue Models for in-transit Analysis

In-transit analysis provides a useful abstraction for separating data capture/use and analysis, enabling different actors (i.e. service providers) to be involved in each of these processes. Hence, data capture may be carried out by a different actor compared to subsequent analysis – enabling multiple capabilities from different actors to be combined at different costs. Each actor may differ in their ability to undertake particular types of analysis that meet varying QoS constraints – leading to different payments that must be made to them by a user to achieve the overall operation.

In our formulation of this problem, we can consider a provider centric view of costs incurred to provide function O. Where a shared Cloud infrastructure is being used, a provider may serve multiple users using a common resource pool through a "multi-tenancy" architecture, or offer multiple functions over their shared infrastructure to one or more users. In both cases, the revenue for the provider is the sum of all the prices $Pr()$ charged to n users for accomplishing m operations O, $\sum_{i=1}^{n}\sum_{j=1}^{m} Pr(O_{ij})$. The provider in turn incurs a cost for performing such operations, $c(O_{ij})$, but can also incur a financial penalty $PSLA_{ij}$ for user i when the QoS targets of operation O_{ij}, identified in the SLA of user stream i are not met. If we assume the objective of the provider is to maximise revenue, then it must decide: (i) which user streams to accept for storage and analysis; and (ii) how many resources (including storage space and computational capacity) to allocate to each stream in order to improve overall revenue (generally over a time horizon). Both of these considerations are based on the SLA that a user and provider have agreed to. By minimising the cost either due to allocation of resources or to SLA penalty, we get the benefit function for the provider as: $\sum_{i=1}^{n}\sum_{j=1}^{m} Pr(O_{ij}) - \sum_{i=1}^{n}\sum_{j=1}^{m} min(c(O_{ij}), PSLA_{ij})$ (Eq. 1)

We consider the SLA for each stream to contain: i) a desired QoS level for each operation, $L_{desired_{ij}}$, ii) the minimum QoS level acceptable to that user $L_{min_{ij}} \leq L_{desired_{ij}}$, and iii) the cost $c(O_{ij})[k]$ for each QoS level L_k defined by the service in the range $[L_{min_{ij}}, L_{desired_{ij}}]$. A provider incurs in a penalty $PSLA_{ij}$, when it fails to meet the minimum level $L_{min_{ij}}$ for O_{ij}. By minimising the cost either due to allocation of resources or to $PSLA_{ij}$, the provider will select an optimal QoS level k_{ij} for each operation O_{ij} specified in the contract such that, (i) the benefit function in Eq. (1) is maximized, and ii) the aggregated number of resources required to provide each operation $O_{ij}[k_{ij}]$ at the required level does not exceed available resources at the provider node.

In general, in order to achieve resource allocation, all resources such as the number of CPUs, disks, I/O bandwidth, buffer sizes and communication bandwidth must be considered. As our QoS guarantees have throughput and delay semantics, only computational resource (number of CPUs or virtual machines allocated to a stream) consumption and buffer occupancy are considered.

3 System Architecture

The resource requirements imposed by each QoS contract level must be known before utility optimization can be made. The utilization of resources are influenced by three components [?,?,?]: (i) the profiling model used for the characterization of the worst-case resource consumption, (ii) the scheduling mechanism used to allocate resources to data streams, and (iii) the accuracy of determining whether there will be enough resources for the aggregated demand. Our system supports the processing of multiple concurrent streams over a shared elastic infrastructure consisting of multiple processing nodes. Each node self-regulates its computing resources to preserve QoS constraints using the three mentioned components and optimizing revenue generation.

3.1 Architectural Components

A node, as depicted in Fig. **??**, involves a combination of traffic shaping, computation and data transfer capability. The *Traffic shaping* component regulates the number of data elements entering into the computing resources, according to the established QoS for each data stream. *Traffic shaping* is based on the use of the Token bucket (TB) model [?]. A TB is characterized by 2 parameters: b and R that are respectively the size of the bucket, and the token generation rate. Tokens are generated and introduced in the bucket at the rate of R tokens/s. Data are stored in the TB buffer until they can be forwarded to the computational resources for processing, based on the availability of one or more tokens in the bucket. When data elements arrive at a rate $r < R$, the generated tokens will build up in the bucket for future usage. In this way, a TB supports bursts of traffic up to a regulated maximum (the amount of data sent cannot exceed $Rt + b$) while isolating data streams from one another and enforcing QoS per data stream. A more detailed description of the TB and its use in this architecture can be found in [?]. The R parameter can be defined to be coincident with the average throughput established for a data stream. For revenue generation at a node, the TB model as an envelop process can be used to estimate cost depending on the resources required during each control period T to process the worst case traffic $RT + b$ for each data stream [?]. However, determining the effective number of computational resources (i.e. a pool of virtual machines at