Topologie frugale pour l'agrégation de flux

Rachid Guerraoui, Erwan Le Merrer, Bao-Duy Tran

Topologie frugale pour l'agrégation de flux

Rachid Guerraoui[1], Erwan Le Merrer[2] et Bao-Duy Tran[3]

[1]*EPFL, Suisse* [2]*Technicolor, France* [3]*Google, Australie*

L'agrégation de flux massifs de données est clé pour l'expansion de l'Internet des objets. En pratique, cette agregation est effectuée le long d'un topologie. Cet article introduit le problème de la *construction frugale de topologies*, destiné au design approprié de topologies pour l'agrégation de flux parvenant à un datacenter. Le traitement de ces flux doit survenir sans perte de paquets, tout en minimisant l'utilisation de machines (ou VMs), appelées *shards*, pour l'agrégation (frugalité). Ce problème apparait tout particulièrement lors du design d'un service à partir de machines fortement contraintes par leur capacité en réception (quantité d'information admissible par unité de temps). Nous présentons tout d'abord le cadre d'application de cette étude, puis les résultats obtenus en termes de quantité d'information véhiculée par les flux de données en entrée des ressources d'agrégation. Nous basant sur ces résultats, nous proposons un problème d'optimisation à résoudre pour construire une topologie de noeuds d'agrégation qui évite les pertes et assure la minimisation du coût d'opération. Nous proposons enfin un algorithme pour la résolution de ce problème et analysons ses propriétés. Cette étude trouve une application directe par l'utilisation d'Amazon Kinesis, une plateforme permettant le traitement de flux massifs de données à destination du nuage.

Real-time computation over flows of data is made possible by the progress of *stream processing* platforms like Storm. Algorithms processing those streams require small space and update time [PGD12, HL14]. The support for *stream aggregation* operations is necessary for advanced analytics since it allows more advanced applications like identification of heavy hitters or anomaly detection. For scenarios like device monitoring, the exploding number of those devices cause technical challenges. Clearly, no single server can sustain millions of concurrent connections for aggregating the data they produce. In systems where computation precision is essential, computing units are organized as a *topology* so that the information received in the cloud is processed and reduced layer by layer in a scalable fashion, until the desired result is obtained [PGD12].

Current Situation : Trial-and-Error Design Today, Amazon Kinesis is the cloud-based solution to aggregate streams in real-time. This service allows data buffering in the cloud, which is a required step before actual processing, to avoid data loss. Kinesis imposes arbitrary limits on the amount of data devices can push to a so-called *shard*. In practice, packets that cause rate exceedance at a given point in time are simply dropped (i.e. lost). Billing is made on the number of shards used by an application. Unfortunately, using such a service at a large scale is challenging. Indeed, there is no systematic way to design a topology for handling all device streams, other than by *trial and error* [PGD12, HL14], where system administors incrementally reinforce parts of the topology where loss occurs. In fact, a working solution obtained through trial and error is likely to over-provision the number of shards (one can for instance provision a tangibly high number of shards, resulting in no data loss). Operational costs will be more than actually required by the application, which is a clear problem when considering the general will to consolidate costs by using the cloud. At the other extreme, if one designs a topology that is "too small" to ingest the streams, device packets will be dropped.

This paper addresses the systematic design of a suitable topology of shards, so that streams can be aggregated with no data loss and at a minimum operational cost. In the meantime, we also control the topology diameter, for allowing to forecast end-to-end aggregation latency.

1 The Problem of Efficient Aggregation in the Cloud

This paper addresses computations over data streams emitted from monitored sources. This for instance includes ISPs' home-gateways, for troubleshooting or performance assessment. To cope with the amount of

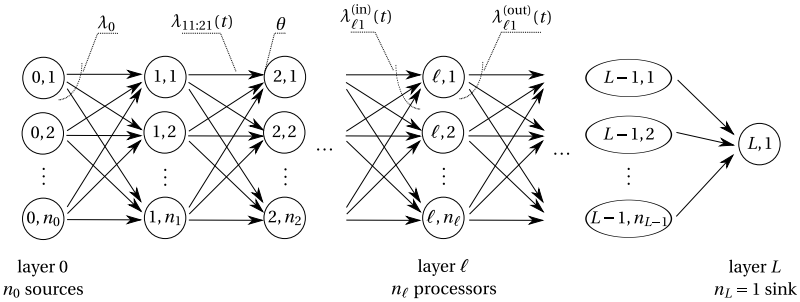

FIGURE 1: General topology form for stream aggregation. Instantiation in practice often has the form of a "pyramid".

data arriving at the datacenter (where the monitoring is performed), there is a need for buffers (*i.e.* Kinesis shards) to temporarily store data before processors can catch up. Examples of buffering systems include Apache Kafka (platform to deploy), and Amazon Kinesis (available as an online service) that we focus on.

Kinesis accepts data records from producers (devices) and releases them to consumers upon request (aggregation processors in our case). Those devices push information to the cloud on a Kinesis interface, implemented with multiples shards where data is actually stored temporarily before processing. Every device pushes data employing a 'partition key', used to direct data records to a specific shard (deterministically assigned by hashing the partition key). Most importantly, Kinesis shards can be chained in a *topology*, to realize more complex stream processing scenarios. In the sequel, a *node* refers to a **shard-processor** couple ; it is the basic unit of buffering/computation, from which aggregation topologies are built.

Considered setup This paper specifically examines stream aggregation over non-overlapping windows [KWF06] : we employ a model where data sources emit streams of key-value tuples (T, v) where T is a time period ID and v is a payload value, compatible with an aggregate function. This allows queries like "`return SUM(v) in T`" for operations over distributive and algebraic aggregates [KWF06]. Such functions permit partial accumulation and parallelization of the aggregation process, to face massive aggregation needs.

Consider the general purpose topology presented on Figure 1 ; n_0 data sources (*i.e.* monitored devices) emit tuples to the cloud, namely to the first layer of aggregation nodes $1,1$ to $1,n_1$. Those nodes aggregate v values to decrease stream size, and in turn emit aggregate values to their successors in the topology, nodes $2,1$ to $2,n_2$ and so forth an so on. Eventually, a final aggregation node named the sink has enough capacity to receive streams from previous layer $L-1$, and then computes the final aggregation value. In this work, we assume that nodes maintain the counter of the current value they are aggregating on. When a tuple containing a new T is received at a node, that node emits the aggregate value to its successor in the topology. We allow maximum two periods to overlap (*i.e.* two values of T), for handling clock drift on nodes and network/processor congestion. In this setup, an ISP for instance can obtain every $T = 5$ minutes, the aggregate of the network traffic produced by all its n_0 monitored gateways, where those gateway emit their current network packet count v every 30 seconds.

The thrifty topology construction problem *Consider the general form topology presented on Figure 1, consisting of n_0 sources emitting tuples at a rate of λ_0, and aggregation nodes organized in L layers and a last aggregation sink, with an ingest rate θ (*e.g. *MB/s) constraint at each aggregation node. Derive a systematic approach to determine a suitable topology such that ingest rate saturation is guaranteed to be averted at all nodes while minimizing the topology size N for reduced cost and operation latency.*

2 Thrifty Topology Construction

2.1 Topology construction as an optimization problem

In order to describe our problem in a more formal way, a mandatory result is the incoming data rate at each node in the topology, as far as we are given a hard constraint on what each node can receive (constraint θ). Following the setup presented in previous section, we precisely derived bounds for incoming and outcoming rates occurring in a topology of the form of Figure 1, at each node. Please refer to report [Tra14] for detailed

Algorithm 1: Conservative Sequential Assignments : $\mathcal{CSA}$

Input: $n_0 \in \mathbb{N}^*$ (sources), $\lambda_0 \in \mathbb{R}^+$ (emission rate at a source), $\theta \in \mathbb{R}^+$ (input rate constraint at a node)
Output: $L, n_1, n_2, \ldots, n_L \in \mathbb{N}^*$ (*i.e.* a topology setup)

for $\ell \leftarrow 1$ **to** $+\infty$ **do**
  $n_\ell \leftarrow \left\lceil \frac{n_0\lambda_0}{\theta 2^{\ell-1}} \prod_{k=1}^{\ell-2} (1 + \frac{1}{n_k(2n_{k+1}-1)}) \right\rceil$;
  if $n_\ell = 1$ **then**
    $L \leftarrow \ell$; **return** $L, n_1, n_2, \ldots, n_L$;

analysis ; we only report the main result, that is the bound on the incoming rate.

Theorem 1. *Total incoming rate at any node* (ℓ, m) *(*$\ell \in \{1..L\}, m \in \{1..n_\ell\}$*) is upper bounded by :*

$$\lambda_{\ell m}^{(in)}(t) \leqslant \lambda_{\ell,\max}^{(in)} = \frac{n_0\lambda_0}{2^{\ell-1}n_\ell} \prod_{k=1}^{\ell-2} (1 + \frac{1}{n_k(2n_{k+1}-1)}), \tag{1}$$

with n_k being the number of processor in layer k. Our goal is thus to avoid data loss in each topology layer, by having a sufficient number of nodes aggregating tuples in each layer : let function $\psi_\ell(n_1, n_2, \ldots, n_\ell)$ encode the fact that capacity of layer ℓ is over the received flow of information at ℓ (*i.e.* no loss ; details in report [Tra14]). We now can formulate the topology construction objective, that our algorithm will solve :

$$\begin{cases} \min\limits_{L, n_1, n_2, \ldots, n_{L-1}} \left(\beta \sum\limits_{\ell=1}^{L-1} n_\ell + \gamma L \right) \\ \text{subject to : } L, n_1, n_2, \ldots, n_{L-1} \in \mathbb{N}^* \\ \text{subject to : } \psi_\ell(n_1, n_2, \ldots, n_\ell) \leqslant 0 \quad (\forall \ell \in \{1..L\}) \end{cases} \tag{2}$$

Note that for topology design, we only need to specify the number of nodes per-layer, as their role is totally symmetrical ; this is due to the random routing of tuples resulting from the hash function used. Parameters β and $\gamma \in \mathbb{R}^{*+}$ are used to linearly combine the two objectives, namely the number of nodes in the topology (left part) and the diameter of it (right part).

Topology determination now boils down to finding a plausible assignment for $L, n_1, n_2, \ldots, n_{L-1}$: the problem (2) is an optimization problem with L integer variables $(L, n_1, n_2, \ldots, n_{L-1})$, L linear range constraints (all variables being positive), L non-linear inequality constraints, and a linear objective function.

2.2 Proposed Algorithm : Conservative Sequential Assignments ($\mathcal{CSA}$)

To satisfy function ψ, appearing as a constraint in (2), the number of nodes n_ℓ in each layer ℓ must satisfy :

$$n_\ell \geqslant \frac{n_0\lambda_0}{\theta 2^{\ell-1}} \prod_{k=1}^{\ell-2} (1 + \frac{1}{n_k(2n_{k+1}-1)}). \tag{3}$$

The criterion for n_ℓ at layer ℓ then *only* depends on the number of nodes at layers *preceding* ℓ. This makes *sequential assignments* of n_ℓ with incremental ℓ possible, layer after layer, starting at front layer 1. The minimization of objective $\sum_{\ell=1}^{L-1} n_\ell$ can then be achieved by *conservatively* selecting the smallest possible value for every n_ℓ. This corresponds to picking n_ℓ as the ceiling of the result of formula (3) for a given ℓ. The process terminates as soon as $n_\ell = 1$ at some ℓ (*i.e.* the sink). This corresponds to the Conservative Sequential Assignment Algorithm, (denoted as $\mathcal{CSA}$), and proposed in Algorithm 1 as a solution for the thrifty topology construction problem.

Analysis of $\mathcal{CSA}$ We first express that Algorithm 1 converges, and thus outputs a topology solving the problem of thrifty topology construction. We finally give an approximation ratio for it.

Theorem 2. *$\mathcal{CSA}$ converges with* $\lim_{\ell \to +\infty} n_\ell = 1$.

Sketch. We show that n_ℓ decreases monotonically until reaching stop condition at $n_\ell = 1$. □

We next show that $\mathcal{CSA}$ terminates in a logarithmic number of steps with regards to input parameters. As a consequence, the resulting topology also has a logarithmic diameter.

Lemma 1. *$\mathcal{CSA}$ outputs a topology whose diameter is logarithmic in the problem input values n_0, λ_0 and θ. In particular, $\mathcal{CSA}$ terminates in at most* $L_{\max} = \left\lceil \log_b \frac{\eta\theta}{n_0\lambda_0} \right\rceil$ *iterations, with* $\eta = \frac{49}{72}$ *and* $b = \frac{7}{12}$.

Sketch. Condition $n_\ell = 1$ is attained at the latest when $\ell \geqslant \log_{(\frac{7}{12})} \frac{\eta\theta}{n_0\lambda_0} = \tilde{L}_{\max}$. Thus terminates in at most $L_{\max} = \lceil \tilde{L}_{\max} \rceil$ iterations. □

This bound on the diameter is crucial for service implementation, in order to estimate for instance the end-to-end operation latency (between a message sent from a device and the aggregation result at the sink).

Let $N = \sum_{\ell=1}^{L} n_\ell$ be the total number of processors in the topology. We now bound N resulting from $\mathcal{CSA}$.

Corollary 1. *$\mathcal{CSA}$ outputs a topology composed of N processors :* $N \leqslant \sum_{\ell=1}^{L_{\max}} \left\lceil \frac{n_0\lambda_0}{\eta\theta} b^\ell \right\rceil$.

N thus indicates operational costs, as in Kinesis user is billed on the number of shards (nodes) employed. We finally give the approximation ratio of $\mathcal{CSA}$:

Theorem 3. *An approximation ratio of $\mathcal{CSA}$ is* $2.06 \times \mathcal{OPT}$*, plus* $\log_b \frac{\theta}{n_0\lambda_0} + 1$.

Sketch. Clearly, no algorithm can use fewer nodes than $\lceil (n_0\lambda_0)/\theta \rceil \leqslant \mathcal{OPT}$. We compare N obtained from $\mathcal{CSA}$ to $\mathcal{OPT}$. Note that this lower bound is not a solution to the problem. □

3 Conclusion

We implemented and experimented $\mathcal{CSA}$ and the following real scenario : $n_0 = 500$ devices continuously send 50KB packets at a rate of $\lambda_0 = 0.5$ packets per second to Kinesis ($\theta = 20$ items/sec) where aggregation is performed on some counters. $\mathcal{CSA}$ returned a topology of the form $\begin{bmatrix}13 & 7 & 4 & 2 & 1\end{bmatrix}$, that lead to no measured data loss. For instance, a trial-and-error example with a $\begin{bmatrix}13 & 7 & 1\end{bmatrix}$ (*i.e.* just few nodes less) topology causes 2% of packet drops at the sink node (most critical packets, as they were the aggregate of many previous tuples). This validates our model.

To conclude, we exposed that systematic approaches to design topologies for data aggregation in the cloud are key for scalability and operational costs. Despite the fact we elaborated on the general form of an aggregation topology, the $\mathcal{CSA}$ algorithm is of course tied to the presented setup. We nevertheless expect this initial step to trigger research works for generalization, and integration into cloud processing platforms.

Références

[HL14] Qun Huang and Patrick PC Lee. LD-Sketch : A distributed sketching design for accurate and scalable anomaly detection in network data streams. In *INFOCOM*, 2014.

[KWF06] Sailesh Krishnamurthy, Chung Wu, and Michael Franklin. On-the-fly sharing for streamed aggregation. In *SIGMOD*, 2006.

[PGD12] Odysseas Papapetrou, Minos Garofalakis, and Antonios Deligiannakis. Sketch-based querying of distributed sliding-window data streams. In *VLDB*, 2012.

[Tra14] Bao-Duy Tran. Systematic approach to multi-layer parallelisation of time-based stream aggregation under ingest constraints in the cloud. In *Master Thesis*. EPFL, 2014. http ://www.erwanlemerrer.fr.eu.org/tran-thesis.pdf.