Streaming Analytics with Adaptive Near-data Processing

Atul Sandur*
University of Illinois at Urbana-Champaign
Urbana, IL, USA
sandur2@illinois.edu

ChanHo Park
UNIST
Ulsan, South Korea
pch8286@unist.ac.kr

Stavros Volos
Microsoft Research
Cambridge, UK
svolos@microsoft.com

Gul Agha
University of Illinois at Urbana-Champaign
Urbana, IL, USA
agha@illinois.edu

Myeongjae Jeon
UNIST
Ulsan, South Korea
mjjeon@unist.ac.kr

ABSTRACT

Streaming analytics applications need to process massive volumes of data in a timely manner, in domains ranging from datacenter telemetry and geo-distributed log analytics to Internet-of-Things systems. Such applications suffer from significant network transfer costs to transport the data to a stream processor and compute costs to analyze the data in a timely manner. Pushing the computation closer to the data source by partitioning the analytics query is an effective strategy to reduce resource costs for the stream processor. However, the partitioning strategy depends on the nature of resource bottleneck and resource variability that is encountered at the compute resources near the data source. In this paper, we investigate different issues which affect query partitioning strategies. We first study new partitioning techniques within cloud datacenters which operate under constrained compute conditions varying widely across data sources and different time slots. With insights obtained from the study, we suggest several different ways to improve the performance of stream analytics applications operating in different resource environments, by making effective partitioning decisions for a variety of use cases such as geo-distributed streaming analytics.

CCS CONCEPTS

• **Information systems** → **Stream management**; • **Computing methodologies** → **Distributed algorithms**.

KEYWORDS

Datacenter monitoring, Streaming analytics, Wide area network, Query partitioning, Edge computing

*The author is currently at AMD Research.

WWW '22 Companion, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9130-6/22/04…$15.00
https://doi.org/10.1145/3487553.3524858

ACM Reference Format:
Atul Sandur, ChanHo Park, Stavros Volos, Gul Agha, and Myeongjae Jeon. 2022. Streaming Analytics with Adaptive Near-data Processing. In *Companion Proceedings of the Web Conference 2022 (WWW '22 Companion), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3487553.3524858

1 INTRODUCTION

Large amounts of data continue to be generated from many different data sources such as datacenter servers in telemetry systems, Internet-of-things (IoT) devices and other services in geo-distributed settings. Analytics applications strive to process these data streams (i.e., a continuous and unbounded sequence of data items) in real time to extract actionable insights from them, for mitigating performance and reliability issues of the target system.

Recently, we investigated an adaptive near-data processing solution for large-scale server monitoring [12]. Traditional streaming analytics in server monitoring systems operate in a centralized architecture wherein service-level application logs and host-level metrics representing the health of various system resources are transmitted to a stream processor. Alerts are generated if the system is observed to have anomalous behavior. A key challenge here is dealing with potential resource bottlenecks resulting from high network transfer costs due to 10s of PBs per day being generated from hundreds of thousands of servers [10].

Monitoring pipelines can leverage available compute on the data sources (i.e., server nodes) to process data locally, thus reducing the amount of data delivered to the stream processor. However, compute on the data source typically results from resource overprovisioning [11, 13, 14]. Thus, the compute budget is limited to minimize interference with the hosted services. The available budget also varies widely across data sources and over time on each data source. Prior dynamic partitioning techniques such as Sonata [5, 8] can exploit available compute resources on the data source to perform near-data processing of the monitoring query. However, they perform coarse-grained operator-level partitioning wherein an operator is executed on the data source only if all of its ingress data can be processed within the compute budget. Unfortunately, such coarse-grained partitioning is not effective in compute-constrained environments on the data source. Furthermore, query partitioning occurs at runtime by a computationally expensive query planner,

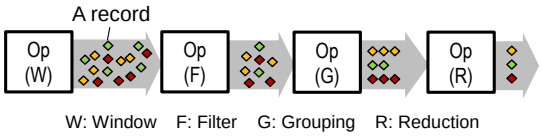

Figure 1: Pipeline of Pingmesh query [12] which probes network latency for node pairs in every time window.

which is unsuitable for making frequent partitioning decisions on data sources which exhibit fast-changing resource conditions.

Our latest work Jarvis [12], presents a partitioning mechanism which performs fine-grained data-level partitioning of the query workload by controlling the amount of data processed by each operator on the data source, while the remaining operator data is processed at the stream processor. Our approach can significantly reduce network data transfers while utilizing the limited and dynamic compute resources on the data sources. The partitioning system is implemented in a fully decentralized manner, so it can scale to a large number of data sources.

The advent of edge datacenters and edge devices have enabled data processing close to the data source in a geo-distributed setup, thus alleviating the network and compute bottlenecks at a central stream processor running in the cloud. Query execution can be distributed between edge and cloud nodes, similar to how we apply Jarvis to datacenter monitoring. However, the resource conditions under which partitioning decisions need to be made can vary widely depending on the considered target system. For instance, datacenter monitoring is constrained by compute resources at the data source, while partitioning for geo-distributed analytics needs to consider scarce and variable bandwidth in a wide-area network (WAN). In this paper, we summarize our major findings from server monitoring systems and present useful guidelines for designing partitioning mechanisms to execute under different resource conditions.

2 COMPUTE-CONSTRAINED QUERY PARTITIONING

2.1 Background & Motivation

The volume of network data transferred in server monitoring systems can be significant. Consider a network monitoring scenario in Pingmesh [7]. An agent on each server collects network latency data between node pairs. Latency-sensitive services (e.g., web search) can utilize this data to monitor the network health of their server nodes and generate an alert if more than a predefined proportion of the nodes (e.g., 1%) have probe latencies exceeding a threshold such as 5 ms [7]. With a per-record size of 86 bytes and assuming that a datacenter consists of 200K servers with each server probing 20K other servers using a probing interval of 5 seconds [7], the data generation rate is estimated to be ~512.6 Gbps. Transferring such a high volume of data would strain network capacity and delay the query execution. This observation is corroborated by many existing monitoring scenarios which exhibit *high traffic volume* [10, 15] due to a large number of data sources and diverse data streams.

A monitoring query consists of a pipeline of operators that incrementally reduce data volume, as shown in Figure 1; operators include filtering, grouping, and aggregation for processing structured numerical data (e.g., periodic health metrics) and parsing, splitting,

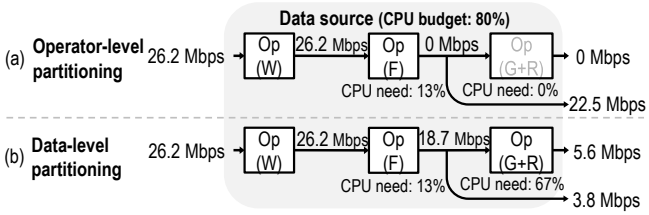

Figure 2: Operator-level (a) vs. data-level (b) partitioning on data source with 80% CPU budget. The total network traffic of (b) is 9.4 Mbps, which is 2.4× lower over (a).

and search for processing unstructured strings (e.g., aperiodic log messages). Network transfer cost can be reduced by partitioning the query and leveraging the *compute available at the data source* to run query operators. Unused compute resources on data sources are typically limited and exhibit temporal variability as resource demands of foreground services change dynamically. For instance, large-scale web (e.g., Alibaba and Wikipedia) and ML inference services hosted on the servers experience bursty request loads in the order of minutes [2, 6]. These services require variable amount of compute to meet their SLAs despite the changing request loads.

Subsequently, query partitioning decisions need to be made promptly *at runtime* on *each data source*, to be compatible with dynamic resource conditions available to exploit on each data source. The ability to quickly adjust query partitioning plans (i.e., *query refinement*) would reduce the time duration for which queries either over-subscribe or under-subscribe available compute resources. Over-subscription can lead to interference with foreground services on the data source, while under-subscription misses an opportunity to further reduce outbound network traffic.

2.2 Design Insights

Combining model-based and model-agnostic techniques for query refinement. Query partitioning problem is NP-hard [12]. Therefore, to efficiently partition a query, we exploit a greedy and embarrassingly parallel heuristic, which utilizes a combination of query cost model-based and model-agnostic techniques. Unlike solving an expensive optimization using an accurate query cost profile, our approach enables data sources to make fast and effective query refinement decisions *independently*. The model-based technique finds a new partitioning plan based on online and fine-grained profiling of each operator's compute cost. However, such profiling might be inaccurate if compute resources are insufficient for accurate profiling. Thus, the model-agnostic step iteratively fine-tunes the output plan produced by the model-based approach.

Data-level query partitioning. Using data-level partitioning, an operator can process a fraction of its input records on data source and drain the rest for remote processing, thus improving the utilization of limited compute resources on each data source.

We highlight the effectiveness of data-level partitioning by performing an empirical study using the query in Figure 1 on a real-world Pingmesh trace. The query is run on a data source node with compute budget set to 80% of a single 2.4 GHz CPU. Our experiment compares the data-level partitioning with the operator-level query partitioning that does *not* allow an operator to process a fraction of

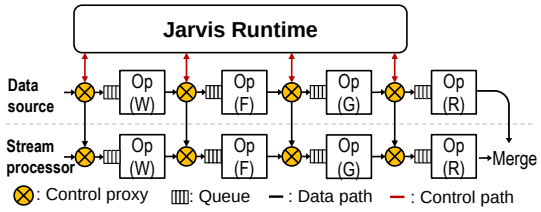

Figure 3: An overview of control proxies and Jarvis runtime.

the input. Figure 2(a) shows that operator-level partitioning cannot execute the costly G+R operator entirely within compute budget as the F operator drops only a small portion of input records. This leads to an insignificant network traffic reduction from the input data rate and leaves the compute budget under-utilized. By contrast, Figure 2(b) shows that the operator G+R can utilize the compute budget fully and process 83% of its input under the data-level partitioning, resulting in a much higher network traffic reduction.

2.3 Partitioning Algorithm

Based on the insights presented in Section 2.2, we propose Jarvis, a new monitoring engine that targets large-scale systems generating high-volume data streams. Jarvis first "replicates" query operators across data source and stream processor nodes, as shown in Figure 3. It then introduces novel extensions to the query execution pipeline: *Control proxy* and *Jarvis runtime*. Control proxy is a lightweight routing logic that bridges two adjacent stream operators and decides "how many" incoming records should be forwarded to the downstream operator on the data source vs. to the replicated operator on the remote stream processor node; the former defines the *load factor* of the control proxy. At each data source, the local Jarvis runtime configures load factors of all control proxies within a query to execute a data-level partitioning plan for the query. The Jarvis runtime continually probes the states of control proxies to observe the query state (i.e., idle, congested, or stable). If the query is in either idle (i.e., all operators are idle for a predefined time duration) or congested (i.e., at least one operator contains pending records more than a threshold) due to changes in resource conditions, Jarvis runtime refines load factors of control proxies to adjust the data-level partitioning plan and keep the query execution stable.

StepWise-Adapt algorithm. *StepWise-Adapt* enables data-level partitioning in Jarvis by using a model-based approach to find a partitioning solution, followed by a model-agnostic mechanism to fine-tune the model-based solution. The model-based approach minimizes network transfer costs due to drained data at control proxies, subject to the available compute budget on the data source. The optimization problem is defined as follows:

$$\min_{p_1,p_2,\ldots p_M} \sum_{i=1}^{M} [\prod_{j=0}^{i-1} p_j r_j](1-p_i) \tag{1}$$

subject to $\sum_{i=1}^{M} [\prod_{j=0}^{i-1} p_j r_j] p_i c_i \leq C/N_r$,

$$0 \leq p_i \leq 1,\ 0 \leq r_i \leq 1,\ c_i \geq 0\ \forall\ i\ \epsilon\ [1,M], p_0 = 1, r_0 = 1$$

Table 1 summarizes the variables used to define our problem. It can be shown that Equation 1 is non-convex and hence it is computationally expensive to solve the problem at runtime. Instead, we identify a transformation (i.e., change in optimization variables) to convert the problem into a linear program (LP). The transformed problem can be solved efficiently using a LP solver. A feasible solution provided by LP solver assumes that operator relay ratios/costs of operators (i.e., r_i and c_i) are fixed and independent of load factors. However, in case the parameters of the optimization problem are not accurately estimated at runtime, the solver provides load factors which would either over-subscribe or under-subscribe the compute budget, making the query execution unstable. StepWise-Adapt takes a second step to address this issue.

Data-level Partitioning Problem	
M	Number of operators in the query.
C	Compute budget available to the query.
N_r	Total number of records injected into the query in a time epoch.
Op_j	j^{th} operator in the query.
r_j	Relay ratio of Op_j, i.e., ratio of its output to input data size.
c_j	Compute cost of Op_j for a single record.
p_j	j^{th} control proxy's load factor, i.e., fraction of incoming records to be processed by downstream operator.
B	Network bandwidth available to the query.
T	Epoch duration in seconds.

Table 1: Variables in the data-level partitioning problem.

In the second step, StepWise-Adapt observes the query state after it executes a time epoch with current load factors of control proxies and fine-tunes them based on the priorities of their downstream operators. Operators are assigned *higher* priority based on if they exhibit *lower* data relay ratio, i.e., lower output to input data size, from operator execution. If the query is in the idle state, StepWise-Adapt then aims to increase the load factor of the operator with highest priority first (until its $p = 1$). On the contrary, if the query is in congested state, StepWise-Adapt then aims to decrease the load factor for the operator with lowest priority first (until its $p = 0$). This approach enables the algorithm to give more resources to operators that potentially result in higher data reduction (and hence higher network traffic reduction). When fine-tuning a load factor, the algorithm executes a binary search over discretized load factor values to further improve convergence time.

Summary of results. Our evaluation of Jarvis shows that a stream processor node can handle up to 75% more data sources while improving query throughput by up to 4.4× over state-of-the-art partitioning strategies, across a wide range of monitoring queries. Moreover, Jarvis converges to a stable query configuration within seven seconds of a resource change occurring on data source.

3 APPLICABILITY

Large cloud platforms today deploy edge servers to process data close to where it is generated, enabling low-latency, intelligent, and real-time geo-distributed analytics [4]. Examples of geo-distributed analytics include (1) CDN log analysis to identify issues related to network bandwidth and latency, to improve CDN data placement [3], and (2) large-scale sensor monitoring in industrial IoT for predictive maintenance [1]. The edge servers, sitting between data sources (over cellular or WiFi) and cloud (over WAN), have compute resources which can be used to identify data points of interest and perform computations such as aggregations and joins as part of a user-defined query. Similar to datacenter server monitoring in Section 2, processing on the edge servers can greatly reduce the outbound network traffic to the cloud connected via WAN. However, WAN bandwidth is very limited and varies significantly over

time [16], requiring stream analytics to operate efficiently under limited and dynamic network constraints.

We now look at how to adapt our compute-constrained query partitioning strategy in Jarvis, for near-data processing in *network-constrained* geo-distributed stream analytics. WAN bandwidth typically sits between 3-20 Mbps and can drop down to below 25% of the maximum bandwidth [16], while compute resources on edge servers can be dedicated for analytics processing. That is, the constraint factor is mainly attributed to dynamic network bandwidth in WAN streaming analytics. Prior work reveals that many WAN-based systems suffer from performance degradation due to high WAN latencies in O(100ms), which result in significant coordination overhead in the system runtime [9]. The decentralized implementation of Jarvis helps avoid this expensive coordination between edge and cloud servers, when making query partitioning decisions.

Now, we change the algorithm in Section 2.3 such that the compute cost of query processing is minimized while subscribing to the available network bandwidth from each edge server. WAN-based partitioning benefits from minimizing the compute cost of query processing on edge servers, because it improves resource utilization of the servers—service providers can then support a larger number of data sources with the provisioned compute resources. While minimizing compute costs, draining data without utilizing compute resources from the servers is not favorable as it can lead to network bandwidth over-subscription. Bandwidth under-subscription is not preferred either due to higher compute costs incurred on the edge servers than necessary. Thus, for a bandwidth B allocated to a query and epoch duration T, we can reformulate the optimization problem in Equation 1 for geo-distributed environments as follows:

$$\min_{p_1,p_2,\ldots p_M} \sum_{i=1}^{M} [\prod_{j=0}^{i-1} p_j r_j] p_i c_i \tag{2}$$

subject to $\sum_{i=1}^{M+1} [\prod_{j=0}^{i-1} p_j r_j](1 - p_i) \leq B.T$,

$0 \leq p_i \leq 1,\ 0 \leq r_i \leq 1,\ c_i \geq 0\ \forall\ i\ \epsilon\ [1, M], p_0 = 1, p_{M+1} = 0, r_0 = 1$

Similar to Equation 1, the new optimization problem can also be transformed and solved efficiently using a LP solver.

Based on the above discussion, we briefly summarize the changes to Jarvis runtime and control proxy to facilitate network-constrained partitioning. In Section 2.3, the query is probed to detect its congestion state and when it is congested or idle, Jarvis runtime refines load factors to keep the query execution stable. For network-constrained partitioning, we instead measure the aggregate network traffic leaving the edge server due to data drained by control proxies and the final query output of the last operator. When the aggregate traffic exceeds a configured bandwidth threshold for the query, Jarvis runtime initiates adaptation. During adaptation, the optimization in Equation 2 is solved in the first step of StepWise-Adapt. For the second step, query state is determined based on the aggregate network traffic leaving the edge server. The load factors are fine-tuned based on the observed state. Operators are assigned higher priority based on if they have lower compute costs during data processing. If the aggregate network traffic exceeds the configured bandwidth threshold, then the load factor of the operator with the lowest compute cost is increased. If the aggregate network traffic is below the given network bandwidth limit, then the load factor of the operator with highest compute cost is reduced.

So far, we have discussed how to extend compute-constrained partitioning techniques in Jarvis to geo-distributed systems with network constraints at the edge servers. We are also studying other extensions such as incorporating energy constraints into the partitioning framework so that near-data processing is done within a given energy budget on each edge server. Such a mechanism will enable monitoring systems to improve the aggregate energy footprint when running streaming analytics.

4 CONCLUSION

In this paper, we study the design and implementation of a fully decentralized data-level query partitioning engine for server monitoring systems. We summarize the insights obtained from the study and then discuss strategies to extend the partitioning framework for geo-distributed analytics systems which can benefit from near-data processing techniques. Finally, we highlight other potential extensions of our framework which could consider energy resource costs when making partitioning decisions.

REFERENCES

[1] AWS. 2022. Industrial Internet of Things. https://aws.amazon.com/iot/solutions/industrial-iot/.

[2] Shuja-Ur-Rehman Baig, Waheed Iqbal, Josep Lluis Berral, Abdelkarim Erradi, and David Carrera. 2019. Adaptive Prediction Models for Data Center Resources Utilization Estimation. *IEEE Transactions on Network and Service Management* 16, 4 (2019), 1681–1693.

[3] Daniel Berman. 2016. CloudFront Log Analysis Using the Logz.io ELK Stack. https://logz.io/blog/cloudfront-log-analysis/.

[4] Matt Calder, Xun Fan, Zi Hu, Ethan Katz-Bassett, John Heidemann, and Ramesh Govindan. 2013. Mapping the Expansion of Google's Serving Infrastructure. In *IMC*.

[5] Tarek Elgamal, Atul Sandur, Phuong Nguyen, Klara Nahrstedt, and Gul Agha. 2018. DROPLET: Distributed Operator Placement for IoT Applications Spanning Edge and Cloud Resources. In *CLOUD*.

[6] Hector Fernandez, Guillaume Pierre, and Thilo Kielmann. 2014. Autoscaling Web Applications in Heterogeneous Cloud Infrastructures. In *IC2E*.

[7] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi-Wei Lin, and Varugis Kurien. 2015. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. In *SIGCOMM*.

[8] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. 2018. Sonata: Query-Driven Streaming Network Telemetry. In *SIGCOMM*.

[9] Fan Lai, Jie You, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. 2020. Sol: Fast Distributed Computation Over Slow Networks. In *NSDI*.

[10] Luo Mai, Kai Zeng, Rahul Potharaju, Le Xu, Steve Suh, Shivaram Venkataraman, Paolo Costa, Terry Kim, Saravanan Muthukrishnan, Vamsi Kuppa, Sudheer Dhulipalla, and Sriram Rao. 2018. Chi: A Scalable and Programmable Control Plane for Distributed Stream Processing Systems. *Proc. VLDB Endow.* 11, 10 (June 2018), 1303–1316.

[11] Rahul Potharaju, Terry Kim, Wentao Wu, Vidip Acharya, Steve Suh, Andrew Fogarty, Apoorve Dave, Sinduja Ramanujam, Tomas Talius, Lev Novik, and Raghu Ramakrishnan. 2020. Helios: Hyperscale Indexing for the Cloud & Edge. *Proc. VLDB Endow.* 13, 12 (Aug. 2020), 3231–3244.

[12] Atul Sandur, ChanHo Park, Stavros Volos, Gul Agha, and Myeongjae Jeon. 2022. Jarvis: Large-scale Server Monitoring with Adaptive Near-data Processing. In *ICDE*.

[13] Weijia Song, Zhen Xiao, Qi Chen, and Haipeng Luo. 2014. Adaptive Resource Provisioning for the Cloud Using Online Bin Packing. *IEEE Trans. Comput.* 63, 11 (Nov. 2014), 2647–2660.

[14] Xiang Sun, Nirwan Ansari, and Ruopeng Wang. 2016. Optimizing Resource Utilization of a Data Center. *Commun. Surveys Tuts.* 18, 4 (Oct. 2016), 2822–2846.

[15] Uber. 2018. The Billion Data Point Challenge: Building a Query Engine for High Cardinality Time Series Data. https://eng.uber.com/billion-data-point-challenge/.

[16] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A. Lee. 2018. AWStream: Adaptive Wide-Area Streaming Analytics. In *SIGCOMM*.