SWAN: WAN-aware Stream Processing on Geographically-distributed Clusters

Won Wook SONG
wsong0512@snu.ac.kr
Seoul National University

Myeongjae Jeon*
mjjeon@unist.ac.kr
UNIST

Byung-Gon Chun*
bgchun@snu.ac.kr
Seoul National University

ABSTRACT

Wide-area stream analytics is commonly being used to extract operational or business insights from the data issued from multiple distant datacenters. However, timely processing of such data streams is challenging because wide-area network (WAN) bandwidth is scarce and varies widely across both different geo-locations (i.e., spatially) and points of time (i.e., temporally). Stream analytics desirable under a WAN setup requires the consideration of path diversity and the associated bandwidth from data source to sink when performing operator task placement for the query execution plan. It also has to enable fast adaptation to dynamic resource conditions, e.g., changes in network bandwidth, to keep the query execution stable.

We present SWAN, a WAN stream analytics engine that incorporates two key techniques to meet the aforementioned requirements. First, SWAN provides a fast heuristic model that captures WAN characteristics at runtime and evenly distributes tasks to nodes while maximizing the network bandwidth for intermediate data. Second, SWAN exploits a stream relaying operator (or RO) to extend a query plan for better facilitating path diversity. This is driven by our observation that oftentimes, a longer path with more communication hops provides higher bandwidth to reach the data sink than a shorter path, allowing us to trade-off query latency for higher query throughput. SWAN stretches a given query plan by adding ROs at compile time to opportunistically place it over such a longer path. In practice, throughput gains do not necessarily lead to significant latency increases, due to higher network bandwidth providing more in-flight data transfers. Our prototype improves the latency and the throughput of stream analytics performances by 77.6% and 5.64×, respectively, compared to existing approaches, and performs query adaptations within seconds.

CCS CONCEPTS

• **Computer systems organization** → **Distributed architectures**; • **Software and its engineering** → **Distributed systems organizing principles**; • **Information systems** → **Data management systems**; • **Networks** → **Network reliability**.

*Corresponding authors

KEYWORDS

Data Processing, Distributed Systems, Networks

ACM Reference Format:
Won Wook SONG, Myeongjae Jeon, and Byung-Gon Chun. 2022. SWAN: WAN-aware Stream Processing on Geographically-distributed Clusters. In *13th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '22), August 23–24, 2022, Virtual Event, Singapore.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3546591.3547524

1 INTRODUCTION

With the surging demand for global services, service providers are increasingly demanding wide-area stream data analytics in order to extract information from the global data generated from multiple distant datacenters [11]. For example, global services need a real-time log processing system for monitoring systems from thousands of distant servers to ensure their SLOs [9, 13]. Also, global services like Twitter [11] need to process distant data in real-time to keep track of global news and social media. Many of such applications often require processing the data with high throughput and low latency as extracting timely information means more value for service providers.

A wide-area analytics system is typically composed of multiple geo-distributed edge clusters and datacenters connected by wide-area networks (WAN) [6, 8, 19, 28]. In this setup, the variability and unpredictable nature of WAN bandwidths make it challenging to achieve both high throughput and low latency. WAN exhibits diverse levels of peer-to-peer (P2P) bandwidths depending on the geo-location, each of which can change in the order of minutes [25, 28]. Essentially, a streaming engine for WAN analytics should be adaptive to these spatial and temporal variability of network bandwidths.

Prior works that consider the spatial variability of networks under wide-area data analytics often focus on short-lived batch processing while reducing network data transfers to lower the network budget and assume that network bandwidth is relatively stable throughout the query execution [17, 22, 24]. On the other hand, existing WAN-aware stream processing systems that run long-running streaming queries try to perform centralized processing after adaptively collecting data to a single data center. Such an approach requires users to trade-off the query output accuracy for performance through pre-aggregation, degradation, and statistical approximation [7, 19, 28], especially when transmitting a large volume of raw data under limited network bandwidths. Despite their great effectiveness, these approaches are frequently application-dependent and may not apply generally. For queries that require high accuracies, such as fraud detection and billing queries, any loss in the query output accuracy may result in undesirable reliability or additional costs. Moreover, determining the right accuracy-performance trade-off typically relies heavily on the expertise of the analyst and requires parameter tuning for each of the different workloads, which may be cumbersome.

To overcome the shortcomings of prior approaches, we investigate a solution that can effectively distribute the query workload over the nodes without loss of query accuracy. In particular, there are multiple ways to distribute the tasks of an analytics workload over geo-distributed computing resources. Our approach for WAN analytics seeks to avoid exercising a path from data source to sink that provides poor bandwidth that comes from very different P2P bandwidths. Moreover, the system aims to adapt the task placement quickly to keep task executions stable despite changing network bandwidths [2, 3, 16]. To achieve the goals, we profile networks to obtain a holistic view on the network path diversity, and keep monitoring the network usage so that more resources could be spent on the networks that need more attention and the system can rapidly adapt to the abrupt changes in resource conditions.

In this paper, we propose SWAN, a new WAN stream analytics engine that achieves the goals by incorporating two key techniques. First, instead of trying to make an ultimate task scheduling solution, SWAN aims to alter the focus on providing a fast solution to keep the latency low, by providing a speed-oriented solution based on a fast heuristic model. Next, SWAN improves the quality of the generated solution by providing more flexibility to task placements through leveraging longer network paths that exhibit higher bandwidths. We have implemented SWAN on Apache Nemo [20, 27] and evaluated it with the NEXMark benchmark suite [21], a popular benchmark suite including multiple queries focusing on different areas of stream analytics. Within seconds, SWAN reduces the average job latency of the queries by 77.6% and increases the throughput rate by 5.64× over using the state-of-the-art distributed query execution.

2 BACKGROUND AND RELATED WORK

2.1 Distributed Streaming Analytics

Query model. We adopt the Apache Beam [1] dataflow programming model to define a query. On our system, a query is expressed as a directed acyclic graph (DAG), where each vertex represents a stream operation, and edges represent the dataflow dependency between the operations. Stream operations that simply process an input record and emit its result to downstream operations (e.g., map, filter) are connected with one-to-one dependencies, while operations that accumulate data from multiple source tasks (e.g., groupByKey, join) require shuffle dependencies ahead of performing the transform. Systems often group the operations connected with one-to-one dependencies as a stage, to pipeline the operations of a task together on a particular machine to reduce the data transfer. These stages are split into parallel tasks to distribute the job to a cluster of multiple machines. Unlike one-to-one dependencies, shuffle dependencies require data transfers over the network from multiple upstream tasks placed on different machines. In an environment with limited network resources, shuffle operations have to occur on the right network in order to prevent the query from suffering network bottlenecks.

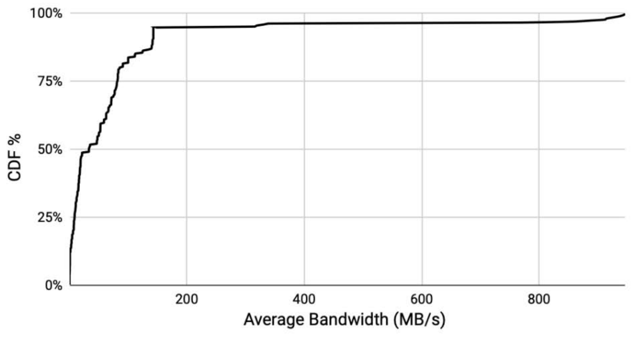

Figure 1: A CDF of networks showing the spatial variation of a geo-distributed cluster.

Stream operator placement. In conventional stream processing systems [4, 5, 12, 18, 26], tasks are generally scheduled in a round-robin fashion for an even distribution of tasks across executors. In order to perform custom task placements on such systems, one must annotate node names on the individual tasks, with the features supported by resource managers. Since existing stream processing systems generally run on local clusters equipped with an excess amount of network resources, they focus on optimizing CPU and memory resources. In order to implement custom scheduling policies, existing systems require modifications on the scheduling layer, as they are often designed with simple support for batch and stream modes for scheduling.

2.2 Streaming on WAN

Existing policies designed for local clusters result in significant performance loss and inefficient resource utilization under WAN, due to the fundamental differences in network environments [17, 22, 24]. While the superfluous network infrastructure on local clusters enables datacenters to reserve network resources for network traffics from particular machines, long-distance cables installed across continents must be shared by multiple different network traffics due to the limited infrastructure. Due to such nature of WAN networks, it exhibits unpredictable variability in both space and time. We characterize the network with two aspects, into spatial and temporal variations, to gain insights on how to optimize streaming engines for WAN environments.

Spatial variability. Spatial variations are caused by the distances and the different network infra connecting the clusters. The infrastructures (e.g., cables, satellites) are usually managed by the internet service providers (ISPs) to connect different LAN networks together. As the infrastructures are each designed with different technologies and budgets, a large diversity exists among the different paths over the WAN. Figure 1 shows the diversity of network bandwidths in a geo-distributed cluster of 16 nodes (i.e., ${}_{16}C_2$ = 120 connections), scattered around 8 different sites over 3 continents (details in § 4.1). Here, we can witness varying average bandwidths as low as about 500KB/s up to 900MB/s, depending on the different locations and distance between the sites, while most network connections show average bandwidths below 100MB/s.

When suggesting a network path between two sites, the ISPs usually provide the path with the lowest latency, but this does not

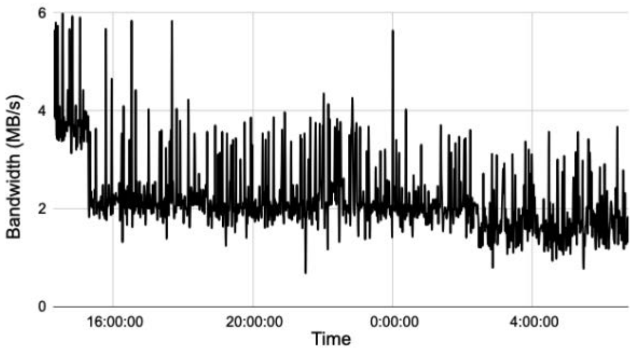

Figure 2: A graph showing the temporal variation of a network through time.

necessarily have the highest bandwidth. If the size of the data to be transferred is larger than the provided bandwidth, it results in network congestion, which causes the latency to rise, as the large traffic has to wait in the queue before being processed. Therefore, it is important to find a network connection that has sufficient bandwidth to accommodate the expected traffic. Subsequently, operator placement decisions need to appropriately made to effectively harvest the available bandwidth across the globe.

Temporal variability. Temporal variations are caused by the high concentration of network traffics through the limited bandwidths of WAN settings. Due to the large number of network users sharing the provided WAN bandwidths, network patterns are highly unpredictable and display diverse patterns over time. Also, the long distance within the WAN adds other physical factors that contribute to the instability of the network. While WAN networks over longer distances often show more instability compared to the ones that are relatively shorter, there also exists heterogeneous variability among the participating regions. WAN network bandwidth patterns can be transient or permanent, and steep or gradual, depending on the cause of the effect, and can occur in bandwidth rises or drops, as shown in Figure 2. Our observations on a geo-distributed cluster show that over half of WAN networks suffer from bandwidth drops of over 20% every 6 minutes on average.

While bandwidth rises do not directly affect the stream analytics performance, bandwidth drops result in increased latency and lower throughput if not handled appropriately. Permanent changes or significant bandwidth drops are particularly considered detrimental and require the system to adapt the task placement to mitigate potential network bottlenecks.

Prior approaches and limitations. Existing data processing systems designed for WAN environments focus on a single type of variation depending on the type of data that they target. Existing systems that target short-lived batch processing jobs [17, 22, 23] focus on the spatial variability to reduce the network data transfers for lower network budget and assume stable network throughout the job.

Regardless, the question of how to perfectly schedule tasks to the geo-distributed nodes while considering all network conditions and task dependencies altogether is known to be NP-hard [14, 15]. Existing works often depend on ILP solvers to schedule tasks in a way that minimizes the longest link transfer finish time of the reduce tasks [17, 22]. Specifically, they optimize the job by observing each stages one after another, to find the right proportion of tasks to place for each node of the geo-distributed cluster. Nevertheless, scheduling tasks using ILP solvers are still significantly slow compared to other conventional scheduling methods despite their simplicity of illustrating the problem [17]. The optimization overhead depends on the length of the query execution plan, but on average, it is 25× slower than conventional scheduling for NEXMark streaming benchmark queries (§ 4.3). Since stream processing has strict latency requirements, it is difficult to adopt an ILP model to effectively handle the temporal variation.

On the other hand, existing systems that target WAN-aware stream processing [8, 19, 28] describe methods to adaptively collect data to a single data center. In order to do so, such systems propose effective ways to pre-aggregate, degrade, and statistically estimate the raw data and trade the output quality for better performance over limited bandwidths. However, while such optimizations are very effective in specific applications (e.g., video processing), accuracy-sensitive applications, such as fraud detection, billing queries, and global stock or transactional analysis, cannot adapt such methods, as lower accuracy can often lead to undesirable reliability and additional problems for such queries [6, 8, 19, 28]. In order to bypass such problems, it is required for the WAN-aware stream processing systems to be designed to run on tasks distributed across a geo-distributed cluster, in a way that can rapidly adapt to the altering conditions.

3 SWAN

3.1 Insights

Good heuristics over an expensive solver. While ILPs provide efficient simple abstractions for materializing the optimization problem of distributing tasks to geo-distributed nodes of a cluster to solve the spatial variability, ILP solvers are too slow to be dynamically used for stream processing systems. ILP could be a good solution if WAN networks do not possess temporal variability and the optimization occurs just a single time. However the 25× overhead is not trivial with job latency if the optimization is to occur repeatedly. In order to mitigate the optimization overhead, we propose using a fast heuristic model that effectively captures WAN characteristics. In building the heuristic model, we aim to put our focus on two primary aspects. First, we aim to find a model that accurately captures the network costs, based on the number of upstream and downstream tasks and the measured network bandwidths, and minimize the network cost throughout the stream analytics job. Second, we try to distribute an even number of tasks to each nodes if possible, in order to prevent computational bottlenecks that can occur when the distribution of tasks are too concentrated on a specific set of nodes.

Query rewriting to fully cover promising longer paths. Wide-area networks are usually managed by ISPs, who, by default, provide the network with the minimum latency upon each request. However, this does not relate to higher bandwidth. Figure 4 shows an example of a network connection between Seoul and Paris that exhibits more network bandwidth if travelled through New York, instead

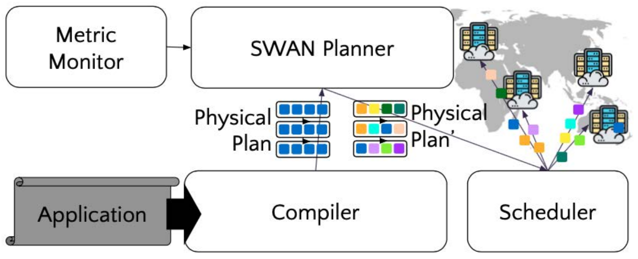

Figure 3: An overall architecture of the SWAN system.

of travelling directly to each other. When provided with a low-latency network with limited bandwidths, the data transfer can suffer from even more latency due to the congestion caused by excess network traffics. Moreover, limited bandwidths lead to full usage of the bandwidth, leaving less room to act as a buffer upon sudden small bandwidth drops with temporal variations. In order to prevent such cases and leverage the network connections with more bandwidths, we perform query rewriting to cover longer paths that are more promising for our workload. In order to capture such network connections, we enable our system to extend the query execution plan with relay operators that simply pass on the data from the uplinks to the downlinks.

3.2 Design Overview

We capture these insights in our system, SWAN, which uses a fast and effective heuristic model for placing tasks on the geo-distributed cluster, with extended optimization techniques to rewrite queries and expand the query execution plan to capture promising longer paths with more bandwidths. Figure 3 shows the key system components involved in the scheduling and optimization of the query execution plan. Once a dataflow application is submitted to SWAN, the compiler performs basic optimizations, such as stage partitioning and determining the number of tasks for the workload, and builds the application into a physical execution plan, which is composed of a group of tasks. With the query execution plan, the SWAN planner collects network metrics and the total number of tasks and their dependencies, to calculate the predicted network cost, and determines where to place each of the tasks. The scheduler takes the physical plan with the placement information and distributes them to the geo-distributed cluster in the way specified by the plan.

When the latency increases in the workload, the metric monitor triggers dynamic optimization, which submits a modified physical plan with the new placement specification to the scheduler. The scheduler fires an optimization mark, which is a special implementation of a watermark that triggers each tasks to checkpoint their data to be migrated according to the new placement specification. Each tasks are sequentially migrated to replay the data from the point of the optimization mark.

3.3 Operator Placement Algorithm

A stream processing application typically has source tasks fixed on specific sites. The tasks of children stages can predict the potential network cost that it incurs if placed on a specific site $s \in S$ among all sites S, based on the number of upstream tasks on the upstream site

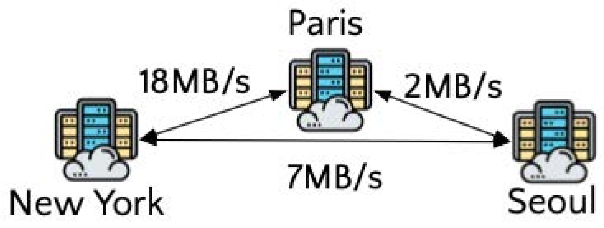

Figure 4: An example of a geo-distributed cluster setup.

$u \in S$ and the given bandwidth between the sites $bandwidth_u^s$. We also find the appropriate number of task slots for each site to evenly distribute tasks among the multiple sites. Based on the calculation of these two values, we distribute tasks according to the ratio of the number of remaining slots divided by the expected network cost of a site. This way, we distribute more tasks to the sites where the network cost is smaller, while also distributing tasks to sites that have more task slots left. We describe the logic above with the following equations:

$$Distribution\_ratio\_factor_s = \frac{task\_slots_s - tasks\_counts_s}{network\_costs_s}, \quad (1)$$

$$s.t.\ network\_costs_s = \sum_{u \in S} \frac{upstream\_tasks\_count_u}{bandwidth_u^s} \quad (2)$$

$$and\ task\_slots_s = \sum_{node \in s} \lceil \frac{\sum tasks\_count}{\sum node\_count} + \frac{1}{2} \rceil \quad (3)$$

To understand the scheduling algorithm, we use an example on a WAN setting illustrated on Figure 4, where three nodes are allocated on each of the three sites, making it a cluster of $\sum node\_count = 9$ nodes. Let us assume a case of executing a three-stage application, where 8 tasks are generating source data in stage 0, followed by 5 tasks in stage 1, and 3 tasks in stage 2. Since the application consists of a total of 16 tasks to be placed on a total of 9 nodes, we set the upper limit for the number of tasks on each node to $\lceil \frac{\sum tasks\_count}{\sum node\_count} + \frac{1}{2} \rceil = \lceil \frac{16}{9} + \frac{1}{2} \rceil = 3$, which results in a total of $3 \times 3 = 9$ slots for each site. Specifying task slots enables us to prevent tasks from being too crowded on a specific site.

Let us assume a case where 3 data source tasks are placed in Seoul and NY, while 2 data source tasks are placed in Paris. In order to schedule the following 5 tasks in stage 1, we first observe the remaining slots for each site, where a total of $9 - 3 = 6$ slots are left in Seoul and NY and $9 - 2 = 7$ slots in Paris. We next calculate the network cost for a potential individual task if it was to be placed on a specific site, described in Equation 2. By this calculation, NY, Paris, and Seoul has the cost of $\frac{2}{18} + \frac{3}{7} \approx 0.5$, $\frac{3}{18} + \frac{3}{2} \approx 1.7$, and $\frac{3}{7} + \frac{2}{2} \approx 1.4$, respectively. With these numbers, we find the target ratio of task distribution as described in Equation 1. This way, NY, Paris, and Seoul has the target distribution ratio of $\frac{6}{0.5} : \frac{7}{1.7} : \frac{6}{1.4} \approx 3 : 1 : 1$. Consequently, 3 tasks of Stage 1 are scheduled on NY, and Seoul and Paris are each scheduled with a single task.

The remaining 3 tasks of stage 2 are scheduled by repeating the steps above. The remaining slots are 3, 6, 5, respectively for NY, Paris, and Seoul. Network costs are $\frac{1}{18} + \frac{1}{7} \approx 0.2$, $\frac{3}{18} + \frac{1}{2} \approx 0.7$, and $\frac{3}{7} + \frac{1}{2} \approx 0.9$ for NY, Paris, and Seoul. According to the logic above, the target distribution ratio is $\frac{3}{0.2} : \frac{6}{0.7} : \frac{5}{0.9} \approx 6 : 3 : 2 \approx 2 : 1 : 0$, and hence 2 tasks are scheduled on NY and a single task is scheduled in Paris.

Our operator placement algorithm can be generally applied to any physical plan consisting of multiple tasks, to place them on an arbitrary number of nodes placed on different sites. In our example, we can observe that NY has the best network conditions among the three sites, and our scheduling algorithm places a total of 8 tasks on NY, while placing 4 tasks each in Paris and Seoul. This way, data can flow into the direction of the site with the most available bandwidths, while preventing a specific site from having too many tasks. While our example illustrates a small example, the slot allocation comes into great usage when the scale of the physical plan grows to hundreds of tasks.

3.4 Query Rewriting

Now that we have a query execution plan with annotations specifying the nodes to place each of the tasks on, we can do further optimizations on the scheduling. On the setting illustrated in Figure 4, we can see that the network between Seoul and Paris exhibit a much narrower bandwidth compared to the bandwidth between two areas through New York. Although we try to use the high bandwidths as much as possible with our operator placement algorithm, a few tasks inevitably have to transfer data through the low bandwidth between Seoul and Paris. In such cases, we provide an extra option to insert a relay task between the tasks in order to be able to send the data through the network going through New York, instead of using the original option.

Nevertheless, if too many tasks transfer data over the high bandwidth, that bandwidth can also be congested due to the large number of traffic. Therefore, we choose to insert the relay task only if the average bandwidth among the tasks transferring data through the network becomes higher with the new option:

$$\frac{bandwidth_{relay\_network}}{\sum tasks\_count_{relay\_network}} > \frac{bandwidth_{original}}{\sum tasks\_count_{original}} \quad (4)$$

Since the major problems in WAN-based analytics occur with lower bandwidths that are intolerable with temporal fluctuations, inserting relay tasks do not significantly increase the latency, as with higher bandwidth, we can allow higher degree of concurrent stream data transfers.

4 EVALUATION

4.1 Methodology

Testbed setup. We deploy our system on 16 GCP Compute Engine e2-standard-4 nodes, each equipped with 4vCPUs and 16GB of memory. We launch 2 nodes on each of the 8 regions on three continents: Taiwan, Mumbai (Asia), Finland, Belgium, Netherlands (Europe), Iowa, South Carolina, and Oregon (America). All nodes run Ubuntu 18.04.

Workloads. We measure the performance of queries from the NEXMark Benchmark Suite [21], a popular benchmark containing a large variation of stream processing queries representing an online auction system. Among the different queries, we spotlight query 4, which calculates for the average price for each category, illustrating

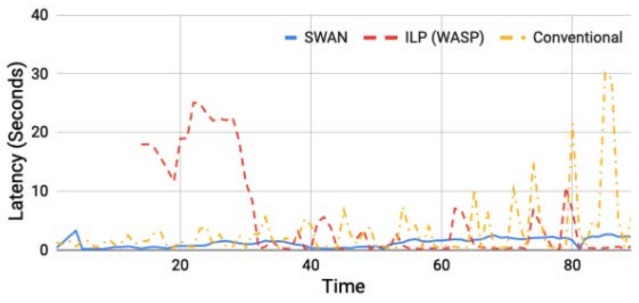

Figure 5: A graph of the 95th percentile latency of the workload after triggering optimization at time = 0 of execution of NEXMark benchmark query 4.

an example of using join and aggregation, which involves shuffle operations.

Prior approaches in comparison. In addition to our implementation of the scheduling policy for operator placement and query rewriting, we also implement prototypes of existing ILP-based solutions described in both Clarinet [22] and WASP [10]. Among the ILP solutions, I have included the ILP solutions that perform better among the two solutions for the results. In order to compare the effectiveness of the heuristic model, we also run the conventional computation-oriented scheduling algorithm for comparison [4, 20, 27].

Performance metrics. We measure the latency and the throughput of the stream analytics for different queries. For measuring latency, we fix the throughput at a fixed input rate and observe the 95th percentile latency changing over time. For measuring throughput, we maximize the input rate, and observe the system performance under the given conditions.

4.2 Throughput and Latency

Figure 5 shows a 95th percentile latency graph comparing the heuristic model with ILP-based model and the conventional scheduling policy over time. In this workload, we set the input rate to 100K events per second, which is about 10MB/s in its actual size. On the graphs, we can see that ILP-based models exhibit the slowest starting time, displaying high latency at the beginning of the workload. As the time goes on, it performs better than the conventional approach, which gradually degrades over time, as it is designed without the consideration of WAN networks. The heuristic model on SWAN exhibits some latency at the beginning of the job compared to the conventional approach, but the latency is negligible compared to ILP solvers. Among the different solutions, SWAN shows the most stable overall latency throughout the workload, displaying its effectiveness for optimizing operator placement on appropriate WAN.

4.3 Query Placement Speed

Figure 6 displays a graph comparing the different approaches of the scheduling algorithms. As described earlier, we can witness a huge overhead, ranging from 20× up to 32× overhead in both ILP cases, compared to the simple heuristic algorithm described in § 3.3. We can witness ILPs suffering from higher overheads with larger query

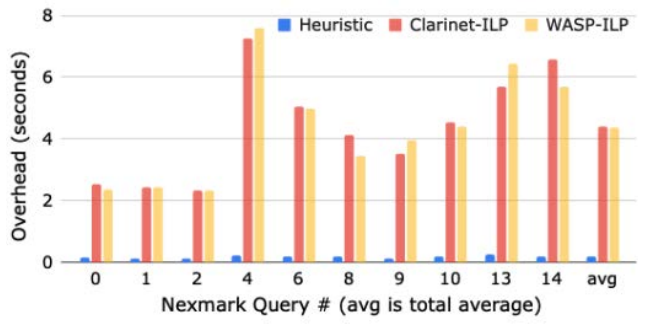

Figure 6: A graph comparing the scheduling overhead of the different approaches.

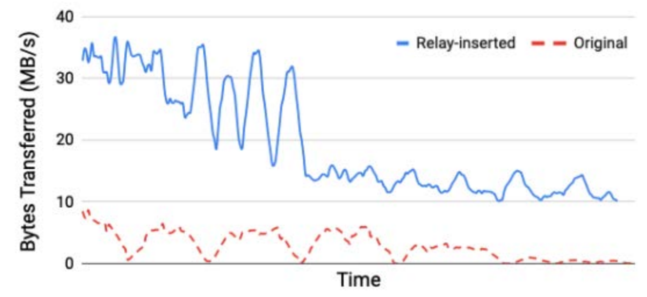

Figure 7: A graph of the throughput of the data transfer rate with and without the relay task insertion in NEXMark benchmark query 4.

execution plans for queries 4, 13, and 14, with $\sigma = 1747ms$ and $\sigma = 1824ms$ each, while the heuristic model exhibits $\sigma = 46.2ms$ throughout the list of different queries.

4.4 Effect of Query Rewriting

In order to spotlight the effect of relay tasks upon query rewriting, we measure the data transfer rate of the workload over time for a workload with inserted relay tasks, and another without the optimization, as shown in Figure 7. We launch the job with maximum input rate, and observe the data transfer rate for the workload. On the graph, we can see that the throughput with the insertion of the relay task shows much better performance compared to the original option. While the throughput rate gradually decreases with the network bottleneck for both options, the relay-inserted workload always displays superior performance for the data transfer compared to the original approach.

5 CONCLUSION

In this paper, we present SWAN, a stream processing system tailored for distributed stream analytics on geo-distributed environments. We point out the problems of spatial and temporal variations that co-exist in WAN settings, and discuss a fast and effective heuristic-based model to solve the problem of scheduling tasks on the different nodes with heterogeneous network conditions in a geo-distributed cluster. In addition, we also discuss a way to optimize the solution further, to add relay tasks appropriately at points where the network can be further optimized by utilizing longer network paths that exhibit more bandwidths, to bypass original supplies of low-bandwidth networks. Our experimental evaluations on latency and throughput show a 77.6% reduction in the average job latency and a 5.64× increase in the throughput rate within seconds.

ACKNOWLEDGMENTS

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2015-0-00221, Development of a Unified High-Performance Stack for Diverse Big Data Analytics).

REFERENCES

[1] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, et al. 2015. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. (2015).
[2] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. 2013. Effective straggler mitigation: Attack of the clones. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. 185–198.
[3] Ganesh Ananthanarayanan, Srikanth Kandula, Albert Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris. 2010. Reining in the Outliers in Map-Reduce Clusters using Mantri. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*.
[4] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015).
[5] Vincenzo Gulisano, Ricardo Jimenez-Peris, Marta Patino-Martinez, Claudio Soriente, and Patrick Valduriez. 2012. Streamcloud: An elastic and scalable data streaming system. *IEEE Transactions on Parallel and Distributed Systems* 23, 12 (2012), 2351–2365.
[6] Benjamin Heintz, Abhishek Chandra, and Ramesh K Sitaraman. 2015. Optimizing grouped aggregation in geo-distributed streaming analytics. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. 133–144.
[7] Benjamin Heintz, Abhishek Chandra, and Ramesh K Sitaraman. 2016. Trading timeliness and accuracy in geo-distributed streaming analytics. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. 361–373.
[8] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. 2018. Videoedge: Processing camera streams using hierarchical clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 115–131.
[9] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. 2017. Pytheas: Enabling Data-Driven Quality of Experience Optimization Using Group-Based Exploration-Exploitation. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 393–406.
[10] Albert Jonathan, Abhishek Chandra, and Jon Weissman. 2020. WASP: wide-area adaptive stream processing. In *Proceedings of the 21st International Middleware Conference*. 221–235.
[11] Kalev Leetaru, Shaowen Wang, Guofeng Cao, Anand Padmanabhan, and Eric Shook. 2013. Mapping the global Twitter heartbeat: The geography of Twitter. *First Monday* (2013).
[12] Wei Lin, Zhengping Qian, Junwei Xu, Sen Yang, Jingren Zhou, and Lidong Zhou. 2016. StreamScope: Continuous Reliable Distributed Processing of Big Data Streams. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 439–453.
[13] Hongqiang Harry Liu, Raajay Viswanathan, Matt Calder, Aditya Akella, Ratul Mahajan, Jitendra Padhye, and Ming Zhang. 2016. Efficiently delivering online services over integrated infrastructure. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 77–90.
[14] Monaldo Mastrolilli and Ola Svensson. 2008. (Acyclic) job shops are hard to approximate. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 583–592.
[15] M MONALDO and S OLA. 2009. Improved bounds for flow shop scheduling. ICALP.
[16] Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. 2015. Making sense of performance in data analytics frameworks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 293–307.
[17] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. 2015. Low latency geo-distributed data

analytics. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 421–434.

[18] Zhengping Qian, Yong He, Chunzhi Su, Zhuojie Wu, Hongyu Zhu, Taizhi Zhang, Lidong Zhou, Yuan Yu, and Zheng Zhang. 2013. Timestream: Reliable stream computation in the cloud. In *Proceedings of the 8th ACM European Conference on Computer Systems*. 1–14.

[19] Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek S Pai, and Michael J Freedman. 2014. Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. 275–288.

[20] Won Wook Song, Youngseok Yang, Jeongyoon Eo, Jangho Seo, Joo Yeon Kim, Sanha Lee, Gyewon Lee, Taegeon Um, Haeyoon Cho, and Byung-Gon Chun. 2021. Apache Nemo: A Framework for Optimizing Distributed Data Processing. *ACM Transactions on Computer Systems (TOCS)* 38, 3-4 (2021), 1–31.

[21] Pete Tucker, Kristin Tufte, Vassilis Papadimos, and David Maier. 2008. *NEXMark—A Benchmark for Queries over Data Streams DRAFT*. Technical Report. Technical report, OGI School of Science & Engineering at OHSU, Septembers.

[22] Raajay Viswanathan, Ganesh Ananthanarayanan, and Aditya Akella. 2016. CLARINET:WAN-Aware Optimization for Analytics Queries. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 435–450.

[23] Ashish Vulimiri, Carlo Curino, Philip Brighten Godfrey, Thomas Jungblut, Konstantinos Karanasos, Jitendra Padhye, and George Varghese. 2015. Wanalytics: Geo-distributed analytics for a data intensive world. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1087–1092.

[24] Ashish Vulimiri, Carlo Curino, P Brighten Godfrey, Thomas Jungblut, Jitu Padhye, and George Varghese. 2015. Global analytics in the face of bandwidth and regulatory constraints. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 323–336.

[25] Hao Wang, Di Niu, and Baochun Li. 2018. Dynamic and decentralized global analytics via machine learning. In *Proceedings of the ACM Symposium on Cloud Computing*. 14–25.

[26] Yingjun Wu and Kian-Lee Tan. 2015. ChronoStream: Elastic stateful stream computation in the cloud. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 723–734.

[27] Youngseok Yang, Jeongyoon Eo, Geon-Woo Kim, Joo Yeon Kim, Sanha Lee, Jangho Seo, Won Wook Song, and Byung-Gon Chun. 2019. Apache nemo: A framework for building distributed dataflow optimization policies. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 177–190.

[28] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A Lee. 2018. Awstream: Adaptive wide-area streaming analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 236–252.