



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NUMBER	FILING or 371(c) DATE	GRP ART UNIT	FIL FEE REC'D	ATTY. DOCKET NO	TOT CLAIMS	IND CLAIMS
62/132,686	03/13/2015		260	14097P		

CONFIRMATION NO. 1984

20844
NEC LABORATORIES AMERICA, INC.
4 INDEPENDENCE WAY
Suite 200
PRINCETON, NJ 08540

FILING RECEIPT



Date Mailed: 03/27/2015

Receipt is acknowledged of this provisional patent application. It will not be examined for patentability and will become abandoned not later than twelve months after its filing date. Any correspondence concerning the application must include the following identification information: the U.S. APPLICATION NUMBER, FILING DATE, NAME OF APPLICANT, and TITLE OF INVENTION. Fees transmitted by check or draft are subject to collection. Please verify the accuracy of the data presented on this receipt. **If an error is noted on this Filing Receipt, please submit a written request for a Filing Receipt Correction. Please provide a copy of this Filing Receipt with the changes noted thereon. If you received a "Notice to File Missing Parts" for this application, please submit any corrections to this Filing Receipt with your reply to the Notice. When the USPTO processes the reply to the Notice, the USPTO will generate another Filing Receipt incorporating the requested corrections**

Inventor(s)

Srimat Chakradhar, Manalapan, NJ;
Naresh Rapolu, Lafayette, IN;

Applicant(s)

NEC Laboratories America, Inc., Princeton, NJ

Power of Attorney: The patent practitioners associated with Customer Number 20844

If Required, Foreign Filing License Granted: 03/27/2015

The country code and number of your priority application, to be used for filing abroad under the Paris Convention, is **US 62/132,686**

Projected Publication Date: None, application is not eligible for pre-grant publication

Non-Publication Request: No

Early Publication Request: No

Title

Dynamic, Network-Award Topology Optimization for Accelerating Stream Processing Applications

Statement under 37 CFR 1.55 or 1.78 for AIA (First Inventor to File) Transition Applications: No

PROTECTING YOUR INVENTION OUTSIDE THE UNITED STATES

Since the rights granted by a U.S. patent extend only throughout the territory of the United States and have no effect in a foreign country, an inventor who wishes patent protection in another country must apply for a patent in a specific country or in regional patent offices. Applicants may wish to consider the filing of an international application under the Patent Cooperation Treaty (PCT). An international (PCT) application generally has the same

page 1 of 3

**MICROSOFT CORP.
EXHIBIT 1006**

effect as a regular national patent application in each PCT-member country. The PCT process **simplifies** the filing of patent applications on the same invention in member countries, but **does not result** in a grant of "an international patent" and does not eliminate the need of applicants to file additional documents and fees in countries where patent protection is desired.

Almost every country has its own patent law, and a person desiring a patent in a particular country must make an application for patent in that country in accordance with its particular laws. Since the laws of many countries differ in various respects from the patent law of the United States, applicants are advised to seek guidance from specific foreign countries to ensure that patent rights are not lost prematurely.

Applicants also are advised that in the case of inventions made in the United States, the Director of the USPTO must issue a license before applicants can apply for a patent in a foreign country. The filing of a U.S. patent application serves as a request for a foreign filing license. The application's filing receipt contains further information and guidance as to the status of applicant's license for foreign filing.

Applicants may wish to consult the USPTO booklet, "General Information Concerning Patents" (specifically, the section entitled "Treaties and Foreign Patents") for more information on timeframes and deadlines for filing foreign patent applications. The guide is available either by contacting the USPTO Contact Center at 800-786-9199, or it can be viewed on the USPTO website at <http://www.uspto.gov/web/offices/pac/doc/general/index.html>.

For information on preventing theft of your intellectual property (patents, trademarks and copyrights), you may wish to consult the U.S. Government website, <http://www.stopfakes.gov>. Part of a Department of Commerce initiative, this website includes self-help "toolkits" giving innovators guidance on how to protect intellectual property in specific countries such as China, Korea and Mexico. For questions regarding patent enforcement issues, applicants may call the U.S. Government hotline at 1-866-999-HALT (1-866-999-4258).

LICENSE FOR FOREIGN FILING UNDER

Title 35, United States Code, Section 184

Title 37, Code of Federal Regulations, 5.11 & 5.15

GRANTED

The applicant has been granted a license under 35 U.S.C. 184, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" followed by a date appears on this form. Such licenses are issued in all applications where the conditions for issuance of a license have been met, regardless of whether or not a license may be required as set forth in 37 CFR 5.15. The scope and limitations of this license are set forth in 37 CFR 5.15(a) unless an earlier license has been issued under 37 CFR 5.15(b). The license is subject to revocation upon written notification. The date indicated is the effective date of the license, unless an earlier license of similar scope has been granted under 37 CFR 5.13 or 5.14.

This license is to be retained by the licensee and may be used at any time on or after the effective date thereof unless it is revoked. This license is automatically transferred to any related applications(s) filed under 37 CFR 1.53(d). This license is not retroactive.

The grant of a license does not in any way lessen the responsibility of a licensee for the security of the subject matter as imposed by any Government contract or the provisions of existing laws relating to espionage and the national security or the export of technical data. Licensees should apprise themselves of current regulations especially with

respect to certain countries, of other agencies, particularly the Office of Defense Trade Controls, Department of State (with respect to Arms, Munitions and Implements of War (22 CFR 121-128)); the Bureau of Industry and Security, Department of Commerce (15 CFR parts 730-774); the Office of Foreign Assets Control, Department of Treasury (31 CFR Parts 500+) and the Department of Energy.

NOT GRANTED

No license under 35 U.S.C. 184 has been granted at this time, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" DOES NOT appear on this form. Applicant may still petition for a license under 37 CFR 5.12, if a license is desired before the expiration of 6 months from the filing date of the application. If 6 months has lapsed from the filing date of this application and the licensee has not received any indication of a secrecy order under 35 U.S.C. 181, the licensee may foreign file the application pursuant to 37 CFR 5.15(b).

SelectUSA

The United States represents the largest, most dynamic marketplace in the world and is an unparalleled location for business investment, innovation, and commercialization of new technologies. The U.S. offers tremendous resources and advantages for those who invest and manufacture goods here. Through SelectUSA, our nation works to promote and facilitate business investment. SelectUSA provides information assistance to the international investor community; serves as an ombudsman for existing and potential investors; advocates on behalf of U.S. cities, states, and regions competing for global investment; and counsels U.S. economic development organizations on investment attraction best practices. To learn more about why the United States is the best country in the world to develop technology, manufacture products, deliver services, and grow your business, visit <http://www.SelectUSA.gov> or call +1-202-482-6800.

Stream processing applications for online analytics are increasingly used in domains ranging from sensor data processing to social networking. To achieve high-throughput, stream processing engines support pipelined execution, low-overhead fault-tolerance, and efficient group communication overlays. The throughput of pipelined application-workflows is significantly impacted by dynamic system state. In particular, we show that a single bottleneck in the pipeline (congested link or an overloaded operator) can drastically impact the system throughput. In this paper, we present a number of techniques for addressing bottlenecks in stream engines. Our techniques fall into two major classes – network-aware routing for fine grained control of streams; and dynamic overlay generation for optimizing performance of group communication operations. To enable fast workflow re-optimization, we present a light-weight protocol for consistent modification of pipelines. We present detailed algorithms, their implementation in a real system, and address issues of fault tolerance and performance. We evaluate the performance of the proposed techniques in the context of three real applications. We show that our techniques improve performance by 20% to 200%, under various overheads, relative to a baseline representative of current implementations. We demonstrate that our techniques are robust to highly dynamic state, as well as complex congestion patterns. Given the widespread use of streaming systems and the need for dealing with dynamic system state, our techniques represent a significant and practical improvement.

1. INTRODUCTION

Stream processing engines (SE) such as Borealis [1], Samza [27], and Storm [25] are commonly used to process continuous streams of data originating from distributed sensors, user-activity logs, and database transactions. Stream processing applications include online learning of complex machine learning models over streaming data [7, 6, 13]. A diverse set of applications have been successfully developed, including click-stream analysis [26], tracking malicious activity (spam classification, intrusion detection), real-time analysis of micro-blogs and tweets [19], and ad-click mining [9].

Stream engines code the application workflow as a directed acyclic graph (DAG) of operators, referred to as a *topology*. Tuples are processed in a pipelined fashion as they traverse through the topology. To achieve exactly-once processing of all tuples and to avoid buffer-overflows in the pipeline, stream engines adopt coarse-grained fault-tolerance and flow-control mechanisms. In this setting, even

a single slow pipeline-stage affects the throughput of the entire pipeline. To sustain high pipeline-throughput over long execution periods, it is necessary to dynamically detect and diagnose any pipeline-bottlenecks.

Emerging online-learning applications have complex topologies and often use structured overlays for group-communication operations. In an online learning application, learner operators process their respective partitions of the input stream (also called an example stream), and update their individual models. Concurrently, learners also synchronize their models periodically using group communication primitives – typically an all-reduce [3]. When compared to traditional streaming workloads, online learning workloads have the following distinct characteristics: (i) models may be large (tens to hundreds of megabytes), which leads to large state transfers between operators; (ii) complex, pipelined group communication (all-reduce) topologies are needed to synchronize potentially large state among all learners. Dynamic orchestration of complex topologies to maintain high throughput in the presence of bottlenecks requires novel techniques.

With the emergence of cloud-computing solutions, stream engines are often deployed on cloud-based virtual machines. Instead of stand-alone deployments, they co-exist alongside other compute and storage systems, such as MapReduce for batch-processing and key-value stores for data. The orchestration of cluster resources among these systems is handled by a global cluster scheduler such as Mesos [11]. In such deployments, stream engines experience network heterogeneity due to several factors.

Co-hosted VM: Two virtual machines (VM) hosted on the same physical machine interfere in their network usage. The bandwidth observed by one VM depends on the traffic flowing into/ from the other VM.

Co-hosted Framework: Cluster managers such as Mesos may allot compute slots to two different frameworks – batch-processing framework such as MapReduce, and a stream-processing framework – on the same physical machine or VM. This leads to network interference and un-even bandwidth availability to the different frameworks.

Co-hosted Topology: Even if only a stream processing system occupies all slots on the entire physical machine, multiple stream-topologies may be scheduled on these slots leading to network interference.

Furthermore, as SEs are deployed for long time periods, workload variations are commonly observed (e.g., activity of energy-measuring sensors exhibits temporal variation), lead-

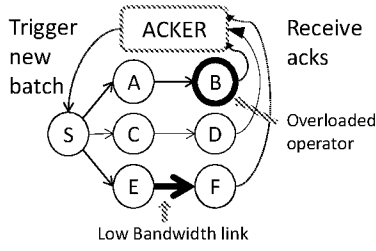


Figure 1: Flow control and fault tolerance mechanism in Storm

ing to temporal skew in CPU utilization. Thus, for efficient stream processing, stream engines must effectively diagnose any pipeline bottlenecks induced by heterogeneity in compute and network resources.

Schedulers built for traditional SEs [17, 31] do not effectively handle new challenges posed by deployment and usage requirements. First, schedulers take a static topology as input. However, for complex group communication operations such as all-reduce, the most efficient overlay structure depends on the network and compute resources allocated to the learner-operators. For this reason, current schedulers are unable to optimize complex communication structures, since they assume that the best topology is known a-priori.

To detect and diagnose temporally varying pipeline-bottlenecks, SEs need a feedback-driven control loop. Furthermore, the diagnosis phase, which involves changing the topology routes, should cause least disruption to the tuple-throughput of the stream engine. Resilient substitution [24] has been proposed recently as a general technique to scale or re-assign topology operators. This is an expensive operation, since it involves multiple steps: stopping the stream, spawning new operators, copying the necessary state from old operator, refreshing network connections, and re-starting the stream. However, frequent invocation of such high overhead methods to react to dynamic bottlenecks is infeasible.

To alleviate these problems, we propose VAYU, a per-topology controller, which uses novel methods and protocols for dynamic, network-aware routing and on-the-fly topology modifications. We rely on three novel techniques to achieve network-aware routing: (i) representing topology link structure using *route-maps*; (ii) consistent hashing for fine-grained key-space management and routing of tuples; feedback information about resource bottlenecks is translated to key-space mapping; and (iii) a light-weight, fault-tolerant protocol for atomic route-map update. By applying novel heuristics on the topology performance (feedback) metrics, the controller determines efficient route-maps, which encode tuple-routing information and also the topology link structure. These new *route-maps* are atomically injected into multiple operators, on-the-fly, using a light-weight, fault-tolerant protocol, for fast topology re-optimization.

We implement our algorithms and protocols in Storm [25]. In the context of three real applications, we demonstrate that VAYU achieves significant performance improvements, 20% to 200% depending on the bottleneck. Furthermore, we show that our improvements are robust to highly dynamic network state, as well as complex congestion patterns.

2. MOTIVATION AND OVERVIEW

We start by describing a low-overhead method for flow-control and fault-tolerance employed by Storm. We then discuss how pipeline-bottlenecks can severely affect throughput, particularly with the mechanisms employed by Storm. Finally, we mention the shortcomings of traditional schedulers in solving this problem. We use these arguments to motivate our proposed solution.

2.1 Flow Control and Fault Tolerance Mechanism in Storm

In Storm, a topology is a graph whose nodes are operators (spouts and bolts) and edges are virtual connections among operators. For simplicity, assume that every topology has a source operator (also called a spout). Figure 1 shows an example topology. Tuples enter the topology through the spout. For every tuple (tup_s) that enters the topology, Storm keeps track of all the tuples that are emitted by bolts as an after-effect of observing tup_s . The emitted tuples are called descendents of tup_s . When a bolt receives tup_s and emits tup_d , it acknowledges by sending *ack* tuples, one each for tup_s and tup_d , to a system managed *Acker* thread. The *Acker* thread calculates the XOR of all identifiers (extracted from *ack* tuples) that it receives. If the topology is a directed acyclic graph (DAG), a tuple's descendents form a bounded tree. In such cases, the *Acker* thread receives the identifier of every tuple twice, and XOR of all these tuple identifiers will result in a zero. Upon observing a zero, the *Acker* thread marks the source tuple tup_s as successfully processed, and a new source tuple is allowed to enter the topology, leading to end-to-end flow control. If the *Acker* thread does not receive acknowledgements for all descendents within a user-defined threshold time interval, then it forces the source tuple (tup_s) to re-enter the topology, leading to re-processing of the entire descendent tree. This method guarantees at-least-once processing of all tuples.

Building upon the basic technique, more stringent guarantees such as exactly-once processing of tuples can be achieved [25]. To this end, tuples are split into batches with monotonically increasing identifiers. Once the *Acker* thread receives acknowledgements for all descendents created by tuples in a batch, it forces the operators to commit the state created by that batch. During commit, the state is tagged with the identifier of the last observed batch. When a batch is replayed, owing to a fault in the system, the committed batch-identifiers are used to ensure that state is not updated twice by tuples of the replayed-batch. More details about the mechanism can be found in [25].

2.1.1 Drawbacks of the flow-control mechanism

The fault-tolerance mechanism described above has low overhead, since it needs constant space per descendent tree (one variable for XORing all tuple-identifiers in a batch), irrespective of the number of tuples in the tree. However, the flow control aspect of the method is highly sensitive to rate of *ack* emission by different operators processing tuples in the batch. Consider the topology shown in figure 1. Assume that the link between operators E and F has low bandwidth. This leads to low rate of tuple movement in S-E-F path of the pipeline. Consequently, the rate of *ack* emission by F is low. The *Acker* thread emits new batches into the system only after reception of acks from old-batches. Furthermore, a constant number of batches traverse the topology at any time. This method of matching sending-rate to the ack-rate

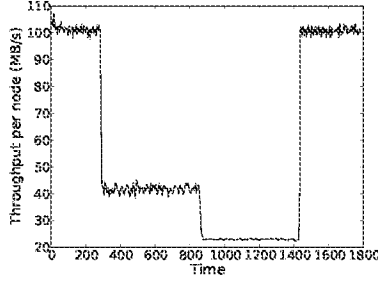


Figure 2: Effect of choking a single random node. Bandwidth choked to 400 Mb/s at 300 sec mark, and to 200 Mb/s at 850 sec, and completely unchoked at 1400 sec mark

is called *ack-clocking* in computer networks [14]. Due to the low rate of ack emission by F, *ack-clocking* ensures that tuples traverse S-E-F path with the rate dictated by the slowest segment (low-bandwidth link). This rate-matching avoids queue overflows and packet drops. However, since the *Acker* thread applies ack-clocking to entire batches, overall batch-emission rate decreases. Consequently, throughput of other pipeline-paths (S-C-D), with potentially high-capacity, also decreases. This leads to low-utilization of resources, and low system throughput. Similar throughput decrease can be observed due to excessive processing delays at an overloaded operator (operator B in figure 1). Note that this phenomenon occurs even if the batch contains only one tuple, but is replicated and forwarded onto different topology-paths. Figure 2 shows the throughput decrease observed in a *tweet hashtag counting* application, when the receiver bandwidth of a randomly chosen node is dynamically varied. In the topology, one set of operators read tweets from external files and passes them to another set of counter-operators. All operators are placed on 16 nodes, ensuring load-balance. The topology throughput drops by more than 50% when bandwidth is reduced from 940 Mb/s (gigabit network) to 400 Mb/s. Also observe that the original throughput is restored when the link bandwidth is restored. This demonstrates how one choked link can cause throughput decrease in the entire topology, due to the coarse-grained *ack-clocking* mechanism.

Maintaining high throughput in the presence of conservative flow-control mechanisms requires dynamic, network-aware re-routing of data to balance load and increase resource (compute and network) utilization.

2.2 Overview of Proposed System

In this paper, we propose light-weight methods for network-aware routing, which is a combination of compute/network load-balancing, along with efficient topology re-optimization. Every operator chooses the destination operator for its outgoing tuples based on *route-maps*. Route-maps contain information on the type and proportion of traffic for each destination operator. The per-topology controller periodically collects metrics from the system. Based on the observed bottlenecks, the controller computes new route-maps that minimize the maximum network and CPU utilization (Section 3). The controller also determines route-maps that tune complex all-reduction topologies (Section 4). The resultant route-maps are installed in a consistent manner on a running cluster, using a light-weight *atomic route-*

update protocol (Section 5). The key insight in the solution is to allow adaptive tuple routing at the sender operators, by sending them feedback information about CPU and network conditions of downstream nodes.

3. DYNAMIC NETWORK AWARE STREAM ROUTING

In a stream-topology, tuples are communicated among operators running on compute nodes. We refer to *grouping* as the pattern of tuple-routing by a set of senders (upstream operators) to a set of receivers (downstream operators). A topology is expressed as a series of groupings between operators. The following are two most common groupings observed in topologies: (i) *shuffle grouping*: upstream operators route each tuple to a random downstream operator; and (ii) *fields grouping*: all tuples with the same values for a given set of tuple-fields – also called a *key* – are routed to the same downstream operator. Fields-grouping is used for aggregation or reduction of tuples with the same key.

3.1 Factors Affecting Grouping Throughput

Several factors affect the grouping throughput, measured as the tuple processing rate of downstream operators.

Network bandwidth skew: The available network bandwidth at downstream operators may be skewed. If the stream is network bound, the queueing delay at the operator with lesser bandwidth would significantly affect the grouping-throughput.

Per-key tuple-count skew: In *fields grouping*, there may be a skew in the number of tuples per key – some keys may be heavy-hitters. For example, in a sensor-processing application, some sensors could be more active than others. Consequently, even if keys are randomly hashed with a good hash function, the downstream operator receiving the tuples bearing the heavy-hitter key becomes a bottleneck.

Current systems, such as Storm, use modulo-based hashing to realize fields grouping. A tuple with key k is sent to the operator with index (i) , where $i = \text{hash}(k) \% m$, where hash denotes the hash function and m is the number of downstream operators. If the number of distinct keys is large, this method leads to good load-balancing among downstream operators. However, in the context of many real applications, the method suffers from several drawbacks:

Inability to accommodate downstream skew: As discussed, *network bandwidth skew* and *per-key tuple-count skew* affect the grouping-throughput. These criteria need to be incorporated in the routing strategy of upstream operators for efficient load-balancing among downstream operators. Modulo-based hashing methods do not allow this flexibility.

Large overhead while scaling: Consider the process of adding a new downstream operator. With modulo based hashing, many keys would be re-mapped to different downstream operators. If downstream operators contain state, key-remapping entails high-overhead state movement between operators.

Lack of specificity: It is not possible to assign a particular heavy-hitter key to a specific operator with more resources.

3.2 Consistent Hashing

To avoid the drawbacks of modulo-based hashing, we use a variant of consistent hashing [16]. Consistent hashing was first used in distributed hash table (DHT) implementations

to accommodate frequent node additions and removals in peer-to-peer systems. In our case, we use consistent hashing primarily to encode the fine-grained information about changing network-capacities and workload imbalance among downstream operators.

In consistent hashing, keys are hashed into a range, say, -2^{31} to $2^{31} - 1$, using a hash function. The range is divided into p contiguous partitions, termed as *buckets*. Assuming there are m downstream operators, initially, each downstream operator is assigned p/m buckets, chosen randomly without replacement. This random assignment leads to load-balance among downstream operators if the bucket count (p) is large and if all downstream operators have equal CPU and network capacities.

3.3 Fine-grained resource assignment

Random assignment of buckets to operators is not sufficient to account for fine-grained *network bandwidth skew* and *per-key tuple-count skew*, in fields-grouping. To this end, the controller periodically collects the following statistic for all buckets in all operators: *per-bucket per-batch tuple-count*, equal to count of tuples received by the bucket in the last batch. It also collects the following system metrics: i) CPU capacities of nodes (measured as millions of instructions per second (mips)); ii) network bandwidths (in and out) of all node-pairs; and iii) system throughput. The controller uses these metrics to increase the pipeline throughput by appropriately assigning the buckets to operators, which are hosted on physical nodes.

Since it is difficult to model the throughput of a complex pipeline, state-of-the-art schedulers aim to load-balance the nodes while decreasing the amount of network traffic [17, 31]. In the same spirit, our controller first balances compute requirements of all pipeline stages by proportionately increasing the cpu-weight (share of total compute-capacity) of cpu-constrained stages. Later, it assigns buckets to operators so as to minimize the maximum CPU and network utilization. For simplicity, assume every node has a replica of all different operator-types. In this setting, a bucket can be assigned to its operator-replica on any node. The resource assignment problem, formulated as an integer programming problem, is an extension of the one used by the COLA [17] scheduler.

3.3.1 Resource assignment problem formulation

For both shuffle and fields grouping, each key-space partition forms a bucket. Let B denote the set of all buckets. If $b \in B$ is a bucket, $D(b)$ denotes the computation-rate (mips) used to process data received by b . Let N denote the set of nodes. For a node $n \in N$, $C(n)$ denotes the computing capacity (mips) of the node. For any assignment of buckets to nodes, we set the decision variable $x_{b,n}$ to 1 if bucket b is assigned to an operator-replica on node n , otherwise it is set to 0. Let $S(n)$ denote the sum of compute-rates of all buckets assigned to n . In terms of the decision variables, $S(n) = \sum_b x_{b,n} * D(b)$. For buckets $b_1, b_2 \in B$, let $F(b_1, b_2)$ denote the rate of data-flow between the two buckets in the stream-topology. Using the decision variable, for nodes $u, v \in N$, the rate of data flowing from u to v , can be defined as $F(u, v) = \sum_{b_1, b_2 \in B} x_{b_1, u} * x_{b_2, v} * F(b_1, b_2)$. Let $R(u, v)$ denote the actual bandwidth between the two nodes.

Controller assigns buckets to nodes (specified by variables $x_{b,n}$) based on the solution of the following integer program:

$$\begin{aligned} \text{minimize} \quad & w_{cu} * \overbrace{\max_n S(n)/C(n)}^{CU} \\ & + w_{nu} * \underbrace{\max_{u,v} F(u,v)/R(u,v)}_{NU} \\ \text{subject to} \quad & x_{b,n} \in \{0, 1\} \forall b, n; \sum_n x_{b,n} = 1 \forall b \end{aligned} \quad (1)$$

The objective of the problem is to minimize a weighted function of two quantities: i) CU: maximum observed CPU utilization; and ii) NU: maximum observed network utilization. The controller also has another competing goal of reducing the total inter-node traffic ($\sum_{u,v \in N} F(u,v)$). Similar to COLA [17], we use a combination of a graph partitioner and load-balancing heuristics to obtain a feasible solution. For large problem sizes (bucket-count is large), graph partitioners are computationally expensive. In such cases, we implement a recently proposed *re-streaming algorithm* for multi-constraint graph partitioning [22], which is shown to be competitive with offline-graph partitioners while using limited resources.

The following are key differences of our formulation with respect to the COLA scheduler: i) modelling traffic as data received by fine-grained key-space buckets instead of coarse-grained operators. This allows fine-grained mapping of buckets to nodes, leading to balanced-load even in the presence of per-key tuple count skew; ii) balancing utilization of all network links is added to the objective function. This is important in environments where network capacity can have significant variations.

3.3.2 Accuracy of network bandwidth estimates

Measuring available network bandwidth between nodes in a cluster already running a stream-processing engine is a challenging problem for two reasons: i) the data-sent by measuring tools, such as IPerf [21], interferes with the stream-traffic, thereby returning noisy bandwidth estimates; ii) due to the interference from measurement-traffic, the stream throughput drops during the measurement-period. To avoid these overheads, the controller mainly uses bandwidth estimates inferred from the rate of tuple-acks emitted by different nodes in the past time-windows. It uses measuring tools for metrics are in-sufficient. Also note that the assignment problem only requires relative bandwidths, instead of accurate values. To avoid frequent bucket re-assignment, controller invokes the control loop only when hysteresis-applied system throughput (rate of tuple-ingestion by the spout) in the recent time windows drops by a threshold percentage (default: 10%) when compared to the long term average. Furthermore, the controller first checks for CPU load imbalance. Once the controller discards the per-key tuple-count skew as the reason for throughput-decrease, it checks for the network bandwidth skew. These policies decrease the usage-frequency of noisy bandwidth estimates, and also helps in setting appropriate weights in the multi-constraint load-balancing heuristics.

The output of the resource assignment problem is a mapping of buckets to operators, referred to as *route-maps*. Using these route-maps, in modified shuffle-grouping, upstream operators choose a random downstream bucket, instead of a downstream operator, when routing tuples.

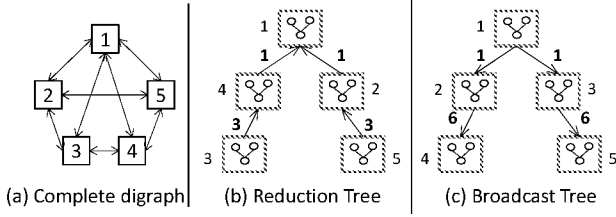


Figure 3: Sample reduction and broadcast trees generated by MWD heuristic

4. TOPOLOGY RE-OPTIMIZATION FOR INTERMEDIATE DATA ROUTING

4.1 Need for group communication operations

Group communications operations, as exemplified by an all-reduce operation, are used by many distributed online learning applications [3, 10]. These operations typically rely on structured static overlays for orchestrating data movement. In this section, we propose an algorithm for determining spanning tree overlays for pipelined all-reduce operations, which explicitly accounts for dynamic network-state.

In an online learning application, operators that train the model using training-examples are termed *learners*. For accurate model training, the model (in our case of a stochastic gradient descent, also called a weight vector) is periodically synchronized among learners, using an all-reduce operation. Static binary trees are among the most commonly used overlays in systems [3, 10] for pipelined all-reduce group communication. Each learner divides its weight vector into slices. Each slice traverses up the tree during reduction and down the tree during broadcast. The tree structure is effective for pipelining slices of a large model or for sending complete models in quick succession, as the network links in a tree allow un-congested traffic flow. However, the throughput of a pipelined tree is heavily influenced by the slowest link. In streaming systems, since model synchronization traffic flows alongside the regular example traffic (input tuples used to train models), the available bandwidth on different links may vary significantly. In this scenario, the tree overlay must be dynamically optimized to suitably use links with higher available bandwidth. Furthermore, in a tree, different nodes (learners) emit their model and receive the reduced model at different times. In this scenario, tree overlays must ensure that low-bandwidth links do not significantly impact model synchronization times for all of the nodes. With these high-level goals, we first define a few terms and formally state the problem. We then describe our method for dynamic computation of efficient overlays.

4.2 Problem formulation

Let G be a complete directed graph, where nodes denote machines hosting learner operators and edges represent potential overlay links. The directed edge-weight between nodes s and d is $W_{s,d}$, which is the time taken to transmit a byte of data from s to d (inverse of link bandwidth). Figure 3 shows a sample reduction tree and a sample broadcast tree generated from the same graph.

Let $t_s(i)$ denote the start-time for model synchronization at node i . This is the time when node i attempts to

reduce its first model-slice with the corresponding model-slices received from its children. Node i subsequently sends the reduced model-slice to its parent in reduction tree. Let $t_e(i)$ denote the end-time for model synchronization at node i . This is the time when node i receives the last reduced model-slice from its parent in broadcast tree. Let $t_{ms}(i)$ denote the model synchronization time for node i , defined as $t_e(i) - t_s(i)$.

Problem Statement: Generate a spanning tree for pipelined all-reduce that minimizes the average model synchronization time, over all nodes.

$$\text{minimize } \left(\frac{1}{|N|}\right) * \sum_{i \in N} t_{ms}(i) \quad (2)$$

In message passing (MPI) systems, overlays are chosen to minimize the maximum completion time of a group communication operation. In contrast, we focus on optimizing the average completion time. This is because, in streaming systems, group communication operations do not follow barriers; they are triggered periodically irrespective of the completion of the previous operations. Furthermore, in online learning applications, average completion time is also an indirect measure of model mixing-rate.

We extend the *Min Weighted Degree Tree* (MWD) heuristic [5] to generate both pipelined broadcast and pipelined reduction trees.

4.3 Pipelined all-reduce overlay generation

To compute a broadcast spanning tree topology, MWD (Algorithm 1) works as follows: Initially the spanning tree contains only the root node. In each iteration, the algorithm adds the least weighted out-edge (intuitively, the fastest out-link), say (u, v) , to the spanning tree. The edge-weights of all other outgoing-edges from node u that are not already in the spanning tree, are incremented by the time node u spends in broadcasting to the previously chosen children. This time is equal to sum of edge-weights to current children ($\sum_{v' \in \text{children}(u)} (W_{u,v'})$). The algorithm continues to select edges until all nodes are included in the spanning tree. When selecting edges, nodes in the spanning tree that have less than a preset threshold k edges are given preference. This parameter can be tuned to generate trees with different branching factors.

To generate spanning tree for all-reduce, we use the MWD heuristic (Algorithm 1) to first find a reduction tree, starting from a given root (v). Since it is a reduction tree, edges coming into the spanning tree (in-edges) are considered when choosing the min-edge. After the reduction spanning tree is generated, its edges are removed from the graph. The algorithm is now run on the residual graph to generate a broadcast tree with the same root node (v). The average synchronization path length, over all nodes, is calculated using the generated reduction and broadcast trees. The algorithm repeats the above steps, each time with a different node as the root of spanning tree. The final chosen root node is the one with the least average synchronization path length, and the final reduction and broadcast trees are the ones generated by the chosen root node. Inside each node, one learner operator is chosen as *leader* and all other learners are connected to it. The leader reduces the model slices generated by node-local learners before sending them to other nodes. It also performs node-local broadcast.

Using fibonacci heaps for edge-set implementation, the algorithm takes $\mathcal{O}(|E| + |V| \log |E|)$ to generate reduction and broadcast trees for a chosen root, where E is the set of edges and V is the set of nodes. The algorithm is invoked $|V|$ times, one for each chosen root node.

Algorithm MWD has a number of desirable properties. The greedy heuristic builds pipelined spanning trees with low weighted out-degree (sum of out-edge weights) of any node in the broadcast tree; correspondingly low weighted in-degree of any node in reduction tree. This strategy minimizes the choking effect of any one stage in the pipeline.

Before generating the broadcast tree, the algorithm removes the edges used for reduction tree. This eliminates the possibility of a single link being used for both reduction and broadcast. The final chosen root is one that reduces the average model synchronization time of all nodes. Intuitively, the heuristic pushes the congested links closer to the leaves than the root, because a congested link close to the root will lie in the synchronization path of a large number of nodes, thereby increasing their synchronization time. In Figure 3 node 5 has low in-bandwidth. Therefore, the heuristic places it among leaves, avoiding its choked bandwidth from affecting the entire pipeline. Furthermore, if node 5 was made the root, it would receive the model-traffic from two children. This extra traffic further decreases the bandwidth for the concurrent training-example traffic, thereby increasing the choking-effect of that node.

A linear tree, generated by setting k to 1, has better pipeline bandwidth than binary tree (k set to 2). Assuming n nodes in total, in linear tree, $n - 1$ nodes are involved in reduction. Even if one of these nodes has less in-bandwidth, it chokes the entire pipeline. Furthermore, for small-sized models, or when the change in model since last synchronization is small, linear tree performs poorly as it takes time proportional to $\mathcal{O}(n)$. On the other hand, binary trees have comparatively less pipeline bandwidth, but offer two benefits: i) since only $n/2$ internal nodes are involved in reductions, ill-effects of choked nodes (upto $n/2$) can be localized by placing them among the leaves; ii) binary trees perform well even for small models due to $\mathcal{O}(\log n)$ height. Therefore, VAYU chooses binary trees by default. Ternary tree (k is 3) can be used when there is considerable skew in node capacities, since only $n/3$ nodes form internal nodes.

5. ON THE FLY TOPOLOGY MODIFICATION

In previous sections, we described methods for dynamically changing key-space assignment and overlay topologies. Another novelty in VAYU lies in the method for updating routing information at operators to reflect the changes proposed by the controller.

5.1 Route Maps as Topology Route Specification

Each operator maintains a *route-map* that specifies the routes on which output messages should be sent after processing incoming messages. For instance, to specify an all-reduce topology, each route-map entry would be of the form: *receive-from* : [1, 2, 3], *send-to* : [4, 5, 6]. This implies, the operator should wait for input message reception from operators with ids: 1, 2 and 3, before sending the output message to operators with ids: 4, 5 and 6. A sample route-map entry

Algorithm 1 Min Weighted Degree Tree

```

1: procedure MIN-WEIGHTED-DEGREE-
   TREE( $V, v_s, E, T, b, TreeType$ )
2:    $TreeEdges \leftarrow \emptyset$ 
3:    $TreeVertices \leftarrow \{v_s\}$ 
4:   for each  $e = (u, v) \in E$  do
5:      $cost(u, v) \leftarrow T_{u,v}$ 
6:   end for
7:   while  $TreeVertices \neq V$  do
8:     if  $TreeType = BroadcastTree$  then
9:        $ReadyVertices \leftarrow \{u \in TreeVertices \mid |children(u)| < b\}$ 
10:       $ReadyVertices \leftarrow \{u \in ReadyVertices \mid$ 
        DISTANCE-FROM-ROOT( $u$ ) is the least}
11:       $link(u, v) \leftarrow \{u \in ReadyVertices, v \notin TreeVertices \mid$ 
         $cost(u, v) \text{ is the least}\}$ 
12:       $TreeVertices \leftarrow TreeVertices \cup \{v\}$ 
13:       $TreeEdges \leftarrow TreeEdges \cup \{(u, v)\}$ 
14:      for each edge  $(u, w) \notin TreeEdges$  do
15:         $cost(u, w) \leftarrow \sum_{v' \in children(u)} (T_{u,v'})$ 
16:      end for
17:    end if
18:    if  $TreeType = ReductionTree$  then
19:       $ReadyVertices \leftarrow \{u \in TreeVertices \mid |children(u)| < b\}$ 
20:       $ReadyVertices \leftarrow \{u \in ReadyVertices \mid$ 
        DISTANCE-FROM-ROOT( $u$ ) is the least}
21:       $link(u, v) \leftarrow \{v \in ReadyVertices, u \notin TreeVertices \mid$ 
         $cost(u, v) \text{ is the least}\}$ 
22:       $TreeVertices \leftarrow TreeVertices \cup \{u\}$ 
23:       $TreeEdges \leftarrow TreeEdges \cup \{(u, v)\}$ 
24:      for each edge  $(w, v) \notin TreeEdges$  do
25:         $cost(w, v) \leftarrow \sum_{u' \in parents(v)} (T_{u',v})$ 
26:      end for
27:    end if
28:  end while
29:  return  $TreeEdges$ 
30: end procedure

```

to specify *fields or shuffle grouping* would be: *send-to*: [10, 344] : [1], [5677, 34345]: [2]. It denotes the assignment of key-space partitions to downstream operators.

5.2 Atomicity Requirements

To change a running topology on-the-fly, route-maps of all involved operators must be updated in an atomic manner – i.e., all nodes must switch to the new route-maps at the same time. For example, if upstream operators route tuples based on different *route-maps*, two tuples with same keys may not reach the same downstream operators, thereby violating the semantics of fields-grouping. Similarly, overlay modification needs to be atomic. If only a sub-set of the nodes have received new route-maps and the other nodes are using the old route-maps, then the resultant topology may not satisfy reduction semantics.

To this end, we propose a light-weight protocol for modifying topology route-maps in an atomic manner. We focus on updating the route-maps with least possible interruption to the existing stream traffic.

5.3 System Constraints

As described in section 2.1, Storm uses a global acknowledgement mechanism to deal with both network-error and operator-failures during tuple traversal.

5.3.1 Fail Fast Operators

Operators in a Storm topology are fail-fast. That is, unlike database nodes that log all their actions into a write-ahead-log, storm operators do not log all input tuples and their corresponding outputs. Only designated operators

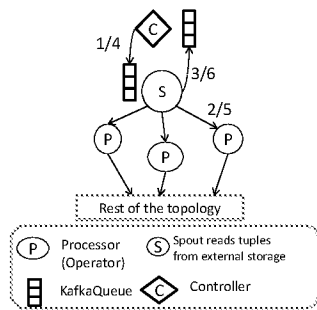


Figure 4: Atomic route-map update protocol

containing state can checkpoint their local state at batch-boundaries. This design helps in quick re-spawning of a failed operator on another node without the overhead of processing any undo or redo logs. In case of operator failure, the global fault-tolerance mechanism ensures re-delivery of unprocessed tuples. This fail-fast design, unfortunately, does not permit the use of traditional atomic commit protocols such as 2-phase or 3-phase commit protocols, which rely on local write-ahead-logs for participant recovery.

5.4 Atomic Route Map Update Protocol

We propose a six-step protocol to assure atomic route-map update. As shown in Figure 4, for every topology, there exist two components: a *controller* (part of the scheduler) and a *spout*; both these components have corresponding kafka queues, which form their message sources. The spout has the following functionality: (i) it truncates the stream into batches and demarcates them by appending *start-batch* and *end-batch* tuples to the stream at batch-boundaries; and (ii) it emits *tick-tuples* to trigger time-based windowed reductions. Truncating the stream into batches permits the use of global fault-tolerance mechanisms. Note that *start-batch* and *end-batch* tuples traverse the entire topology DAG starting from the spout. Tick-tuples are used to periodically trigger all-reduce (or windowed aggregation operations) on all operators.

To update route-maps, the controller creates new route-maps for each involved operator, tags the new maps with a version number (which increases monotonically), and executes the following six-step protocol.

1) Controller first stores the new route-maps in its local state, durably stored in zookeeper. Later, it sends the new route-maps message, tagged by a version-id, to the spout (S), by placing it in the latter's kafka queue.

2) Spout reads the new route-maps from its kafka queue, appends an *install-routes* command to the message, and sends it to all the involved operators by piggybacking on the next *start-batch* tuple. The spout waits for acknowledgements from involved operators; this happens through Storm's *Acker* interface. On reception of route-maps, operators do not immediately switch to the new route-maps; they simply append it to their list of route-maps.

3) After receiving acknowledgements from all operators, the spout sends a *routes-installed* confirmation message to the controller by placing it in the latter's kafka queue.

4) On reception of the *routes-installed* message, the controller durably stores the new topology-route-maps in its local-state. As the controller is part of the scheduler, this local-state is stored in zookeeper. The controller now sends

a *activate-new-routes* message to the spout by placing it in the latter's kafka queue.

5) On receiving the *activate-new-routes* message, the spout first appends the received message onto the next *start-batch* tuple. The controller then waits for the successful commit of all currently executing batches before sending the piggybacked *start-batch* tuple. Since all operators start using the new route-maps for the same batch, semantics of grouping and reduction among operators is consistent.

6) Once the spout receives all acknowledgements for the *start-batch* tuple containing *activate-new-routes* message, it sends an *activated-new-routes* message to the controller, by placing it in the latter's kafka queue. When the controller receives the message, it marks the successful completion of the protocol.

5.5 Correctness of the Protocol

We prove the correctness of the protocol by showing that it does not violate the following safety properties.

No duplicate state changes: Two operators must not update the state for the same key-space bucket in a batch. Our protocol ensures this property since every batch adopts a single route-map version, and all operators operating on a batch follow the same route-map.

Access to complete state: Once an operator assumes ownership of a key-space bucket, it must have access to all the state previously generated for that bucket. This property enables owner operator to process all queries involving keys in its assigned bucket. The protocol satisfies this property as new routes are activated only after ensuring that all previous batches have committed. Therefore, the new owner of the bucket can fetch any needed state from the persistent store while answering queries.

Consistent group communication: For reduction among operators, the spout ensures that all the emitted tick-tuples fall into the same batch. This ensures that all operators follow the same route-maps for reduction.

5.6 Protocol Fault Tolerance

This section describes the mechanisms used by the protocol to handle tuple and operator failures at various steps.

5.6.1 Route Installation Phase

The route installation phase is marked complete only after the controller read the *installed-routes* message from the spout and subsequently stored all the versioned route-maps in zookeeper. If installation fails before this point, controller times-out and retries the installation phase. The controller ensures that the protocol does not move to the *activate-new-routes* phase until the previous *route-installation* phase successfully completes. This ordering ensures that all operators are aware of the new routes before any of them starts to send messages along new routes.

Three types of faults are possible: spout failure, operator failure, or tuple loss due to network failure. If spout fails, then controller times-out and retries the phase. In the event of operator failure or tuple loss, the spout times-out in receiving the acknowledgements, and re-tries the installation phase by piggybacking on the next *start-batch* tuple.

5.6.2 Route Activation Phase

Since route-activation is piggybacked on *start-batch* tuples, any operator failure manifests as a regular topology

failure. The system reacts to operator failure in two ways: (i) the controller re-spawns the operator with all the latest route-maps information. Since controller is in *route-activation-phase* it is guaranteed that all other live operators have the latest route-maps; and (ii) if the spout emitting the stream tuples times-out, it re-emits the batch tuples. Since, every *start-batch* tuple carries the *route-map* version number that the operators are required to follow, each operator will follow the correct route-map during next aggregation/reduction.

5.6.3 Controller Failure

The controller always logs its topology modification related actions in zookeeper. This is done to make it fail-fast. Thus, re-spawning the failed controller is a sufficient fault-tolerance mechanism for the protocol to progress. The re-spawned controller will work-off the state stored in zookeeper. Furthermore, it reads the pending message from its kafka queue. For this reason, it can never miss any messages. Since communication between the spout and the controller always takes place through durable kafka queues, message loss in that communication channel is not possible.

5.6.4 Need for Two Phases

The first phase (route-map installation) is used to ensure two conditions: (i) all involved operators have sufficient resources (memory capacity, connections to new, scaled-out operators, etc.) to exchange buckets as dictated by new route-maps; and (ii) all operators have the new route-maps. Once these two conditions are satisfied, the second phase (route activation phase) can atomically switch to new routes without any system-level interruptions.

6. EXPERIMENTAL EVALUATION

In this section, we present a comprehensive evaluation of the two techniques presented in this paper – dynamic routing for dealing with overloaded network-links/cpu; and adaptive overlays for group communication. The goal is to demonstrate the effectiveness and robustness of the proposed techniques. We conduct all experiments on a 30-node cluster. Each node has a 2.4 GHz quad-core Xeon processor with 8GB of RAM, connected via gigabit ethernet links.

6.1 Static versus Dynamic topologies

6.1.1 Applications

To compare the performance of static and dynamic (network-aware routing) topologies, we implemented three representative streaming applications from different domains: (i) hashtag counting on tweets; (ii) a malicious url detection algorithm [20], representative of an online learning application; and (iii) stream analytics on sensor measurements from DEBS 2014 grand challenge [15].

In hashing counting application, tweets are emitted to random counter-operators which maintain a count of unique hashtags observed in tweets. The application mainly uses shuffle-grouping. In the *malicious url detection* application, the learner operators train a linear model using incoming (shuffled) spam urls from multiple sources: tweets, emails, blacklists, etc. For training, each learner runs regularized

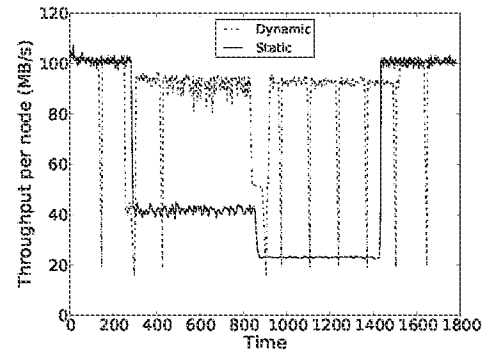


Figure 5: Effect of choking a single random node. Bandwidth choked to 400 Mb/s at 300 sec mark, and to 200 Mb/s at 850 sec mark, and completely unchoked at 1400 sec mark

logistic regression implemented in *vowpal-wabbit* [29] using stochastic gradient descent (SGD). Learners synchronize their models (weight vectors) through all-reduce operations using a spanning tree overlay imposed on the learners [10]. In literature, other online learning applications have reported weight vectors for 20 million features [28]. To study the performance of reduction pipelines on weight vectors of varying sizes, we introduce appropriate random features into the dataset. This application mainly uses shuffle-grouping and all-reduce overlays.

The sensor analytics application [15] addresses analysis of energy consumption measurements from sensors deployed in a smart grid. The *load prediction* query forecasts the energy demand of a smart plug in the near future based on measurements sent by its sensor in a past time-window. Another query detects *outliers* among the smart plugs based on their past usage. Both queries use fields-grouping for time-windowed processing of measurements from a particular sensor. Two issues manifest in a stream-engine that processes measurements from a large number of sensors. The rate of emitted measurements across different sensors is prone to skew: temporal skew occurs when sensors in one continent do not emit observations (during night-time) while sensors in other continents are active; spatial skew occurs when some sensors emit frequent measurements due to heavy usage. Fine-grained tracking of bucket (key-space) load is key to handle throughput loss due to skew.

6.1.2 Effect of Link Congestion on Static and Dynamic Topologies

To test the effect of dynamic network-aware routing, we randomly choose certain nodes hosting counter-operators (in hashtag counting app) and decrease their in-bandwidth using *traffic control (TC)* and *intermediate functional block (IFB)* tools in linux. The controller detects the choked receiver via the metrics interface. Subsequently, the controller creates new-route maps and installs them in the topology.

6.1.3 Impact of Dynamic congestion

In this set of experiments, we investigate the response behavior of the controller when one link-state is dynamically varied. Here, a random learner is chosen and its in-bandwidth choked according to the following pattern: at

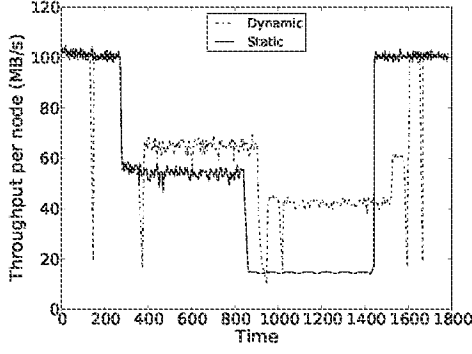


Figure 6: Effect of choking multiple nodes (complex congestion). At 300 sec mark, bandwidths are sampled from Normal(mean=700Mb/s, sd=200Mb/s). At 800 sec mark, bandwidths follow Normal(mean=400Mb/s, sd=300Mb/s). At 1450 sec mark, all nodes are unchoked

the 300 sec mark, the in-bandwidth is choked to 400 Mb/s; at 850 sec mark, it is choked to 200 Mb/s; at 1400 sec mark, it is unchoked to its full gigabit bandwidth. Figure 5 shows the throughput-per-node (volume of input tweets processed per node) as a function of time, for the *hashtag counting* application. Results for the other two applications showed a similar pattern.

For static overlays, once choking sets in, throughput drops significantly (almost 60% for 400 Mb/s and 80% for 200 Mb/s), even though only a single link is choked. In the case of dynamic overlays, once the controller detects throughput loss and new route-maps are incorporated into the topology, we observe a substantial increase in overall throughput. After 400 Mb/s choke, throughput returns to 95 MB/s from 45 MB/s, corresponding roughly to 200% increase in throughput. For 200 Mb/s choke, performance increases by almost 300% when choked link utilization is reduced. Our experiments demonstrate that network-aware routing can be used to recover a substantial part of this lost performance. Note the downward spikes (intermittent loss of throughput), in the dynamic case. They coincide with the times when the controller triggers either network bandwidth measurements or new route-map installation, after observing changes in system throughput. Activation of new route-map requires flushing of current batches to avoid semantic inconsistencies. The loss in throughput for a brief time window leads to more accurate load-balancing and consequent increase in throughput over the long-term. Furthermore, due to hysteresis in measurements, the controller takes a while to react to loss in system throughput. The reaction time of the controller can be tuned by adjusting the hysteresis parameters.

It is important to note that while performance improvement from dynamic, network-aware routing are substantial, it could not completely regain the lost throughput. This is due to two reasons: i) the measure of available-bandwidth at learners is not very-accurate; and ii) the stream-traffic removed from the choked nodes is now processed by the other nodes, along with their own traffic, thereby increasing the total batch-processing delay.

6.1.4 Impact of Complex Congestion Patterns in Link State

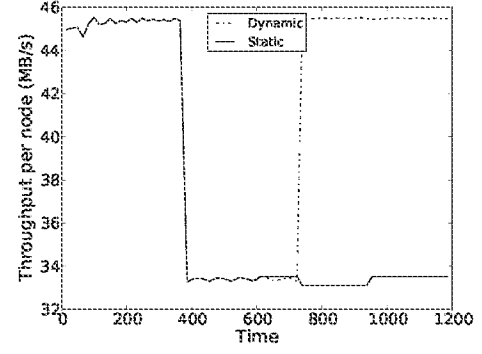


Figure 7: Impact of operator load skew. Initially all sensors emit as same rate (5 tuples/s). At 360 sec mark, sensor emission rates follow Normal(mean=5 tuples/s, sd=5 tuples/s)

To realize complex congestion patterns typically observed in cloud settings when the system runs for long time periods, we choke nodes based on sampling a distribution. Figure 6 shows the performance of static and dynamic topologies under multi-node complex congestion pattern. At 300 sec mark, in-bandwidths are sampled from a *normal* distribution (mean=700Mb/s, sd=200Mb/s). At 800 sec mark, bandwidths follow normal distribution (mean=400Mb/s, sd=300Mb/s). At 1450 sec mark, all nodes are unchoked. As is evident from the results, when the standard-deviation is large (300Mb/s), the improvement in throughput is large (more than 50%). This arises due to the need for accurate load-balancing between slow and fast links. When standard deviation is small, the difference between in-bandwidths is not significant. Consequently, the need for dynamic load-balancing diminishes and the performance gains are commensurately lower.

6.1.5 Impact of skew in per-key tuple count

To test the impact of CPU load imbalance created by per-key tuple count, we use the sensor-analytics application, which relies on fields-grouping for analyzing the pattern of individual sensor measurements over a time-window. We create skew among sensors by sampling their measurement emission rates from a normal distribution.

Figure 7 shows the throughput increase due to fine-grained balancing of tuple-skew by the controller. At the 250 sec mark, the rate of sensor-measurement emission is sampled from a normal distribution(mean=5 tuples/s, sd=5 tuples/s). The small number of nodes hosting the sensors with high rate of emission, process more data, adding extra delay to the batch-pipeline. The controller detects the reduced throughput and triggers load-balancing of the fine-grained key-space buckets, leading to sensor re-assignment. Throughput improves by more than 20% after bucket re-assignment.

The above results show the benefit of fine-grained tracking of per-sensor activity via key-space bucket monitoring. Consistent hashing enables fine-grained key-space partitioning, which is needed to track, diagnose and rectify skewed sensor activity. Selective bucket re-assignment leads to selective-sensor allocation to nodes.

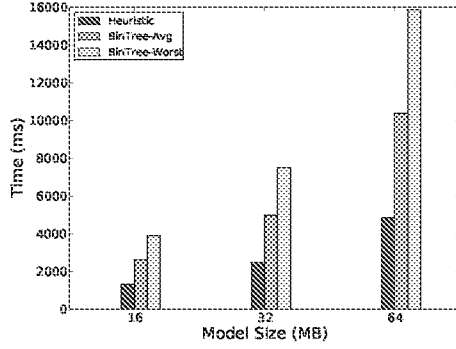


Figure 8: Varying model size. In-bandwidth of a random node choked to 100Mbit/s

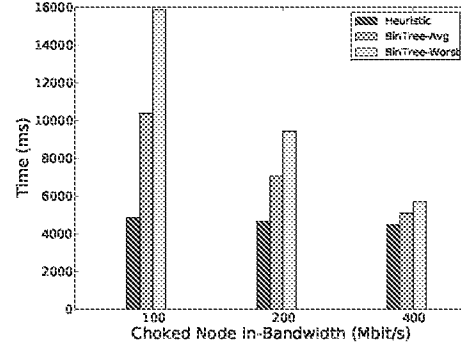


Figure 10: Varying choked node bandwidth. Model size = 64MB. Num choked nodes = 1

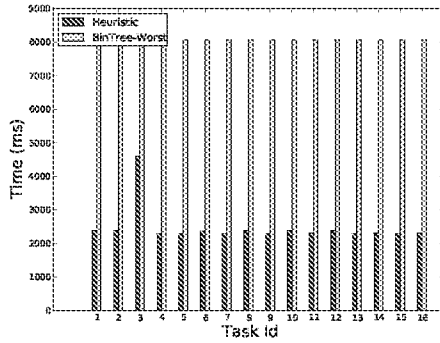


Figure 9: Individual model sync times. In-bandwidth of a random node choked to 100Mbit/s

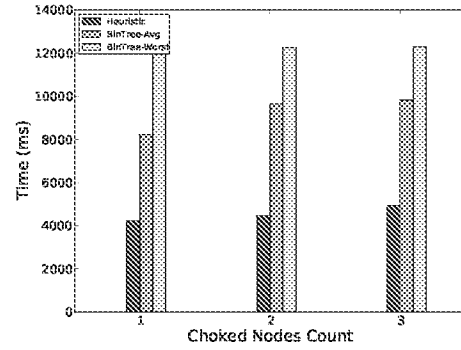


Figure 11: Varying number of choked links. Model size=64MB. Bandwidth choked to 200Mbit/s

6.2 Performance of Dynamic Overlays on Group Communications

In typical learning applications, learners periodically communicate to synchronize their models. In the following set of experiments, we compare the average model synchronization times observed in spanning tree overlays obtained through two techniques: a random, static binary tree (baseline) and the proposed MWD approach. In each case, we quantify the impact of link congestion on performance.

6.2.1 Performance Improvement from Min-Weighted-Degree (MWD) Approach for Varying Model Sizes

Figure 8 shows the impact of choking link bandwidth on average model sync times for different weight vector sizes. In this experiment, the in-bandwidth of a randomly selected node is choked to 100 Mbit/sec. Learner tasks, hosted on 20 nodes, are involved in the all-reduce operation. Our experiments demonstrate that the proposed MWD approach outperforms the random binary tree by a significant margin (more than two-fold speedup), for different model sizes. To further understand this result, we plot the model sync times observed by various learners in Figure 9. In case of MWD, only the latency of the single choked node is affected. This is because the heuristic places the choked node among the leaves of the spanning tree. On the other hand, a random binary tree, in its worst case, can place the choked learner in

the interior of the tree thereby choking a significant portion of the pipeline. In this way, MWD achieves significantly better average synchronization times.

6.2.2 Impact of Varying Link Bandwidth

Figure 10 shows the average model sync times observed for a 32 MB model on 20 learner nodes, with different levels of in-bandwidth choking. It can be seen that as the choking increases, the improvement from our MWD approach increases as well (more than 15% improvement even for 400Mbit/s). This is due to the fact that MWD successfully localizes the lower bandwidth links to the lower levels of the tree.

6.2.3 Impact of Complex Link Bandwidth Patterns

Figure 11, quantifies the effect of multiple choked links. The links are choked to the same magnitude of 200 Mb/s. The weight vector size in these experiments is 64MB. Increase in the number of choked links leads to increase in performance benefits of our MWD approach (more than 30%), when compared to the average-case binary tree. This can be explained as follows: as number of choked links increase, there are more chances of a random binary tree placing one of the choked nodes in the interior of the tree and thereby allowing the choked node to impact the overall pipeline throughput. However, note that the performance of the worst case binary tree, where all choked nodes are placed in the interior of the tree, does not vary substantially. This

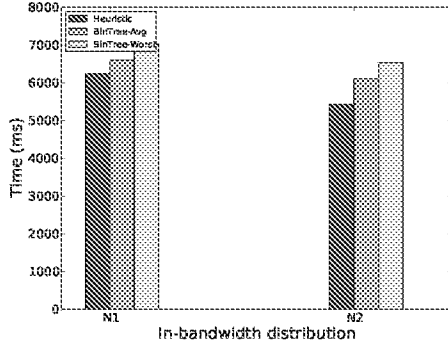


Figure 12: Complex congestion pattern. Model size=64MB. N1 = normal(mean=500Mb/s, sd=200Mb/s), N2 = normal(mean=700Mb/s, sd=300Mb/s)

is because, our implementation divides the model into small parts and sends the parts as separate messages in a pipeline. Furthermore, the rate of the pipeline depends entirely on the slowest link, irrespective of the number of such slow links. However, if the model-size is small, the all-reduce implementation transmits the model as a single message, without any pipelining. In such cases, a random binary tree could place the choked nodes in different levels of the tree, leading to an accumulation of delays. In contrast, MWD places all the choked-nodes among the tree-leaves, ensuring that delays due to choked nodes are overlapped.

Figure 12 quantifies the average model sync time when nodes' in-bandwidths are sampled from two normal distributions: (i) mean is 500Mb/s and standard deviation is 200Mb/s; and (ii) mean is 700Mb/s and standard deviation is 300Mb/s. As evident from our results, when the standard deviation is high, link-bandwidths are dispersed, leading to increased scope for improving the topology. The difference in average sync time is more than 13% between the best and worst overlays for the case of large standard deviation.

7. DISCUSSION

Using large number of buckets leads to fine-grained tracking of key-space activity. However, it also increases overhead of metrics collection. Thus, we plan to extend VAYU with methods for dynamic merging of contiguous buckets when they exhibit similar activity, and methods for splitting buckets when finer-grained tracking is needed.

Current implementation of VAYU does not support dynamic physical resource scaling through node additions. However, the two phase nature of route-map update protocol, provides clear interfaces for such extensions. In particular, the route-installation phase (first-phase) can be used to establish connections with new nodes. Due to the strict ordering between phases, hot-swapping of route-maps in second-phase is guaranteed to maintain correct operation semantics.

8. RELATED WORK

Stream processing systems were initially designed to process continuous database queries on incoming event streams under low latency. These events could be generated by sensor networks in industrial process control, stock quotes,

weather monitoring services, etc. Early stream processing systems include Aurora [2], Borealis [1] and System S [30] from IBM. Recently, several open-source stream processing systems have been developed. Examples include S4 [26] by Yahoo!, Storm [25] by Twitter, and Samza [27] by LinkedIn. These systems have been designed specifically for analyzing clickstream data generated by logging online user activity on websites. Current generation stream-engines focus on enabling high-throughput stream processing along with new features such as: guaranteed message processing, fault-tolerant operator state management, and efficient scale-out mechanisms. Recently, several research efforts have investigated efficient mechanisms for fault-tolerant state management and operator scale-out [18, 8]. In contrast, we focus on heuristics and protocols for fast topology re-optimization, to maintain high stream-throughput at all times.

The TimeStream [24] system uses a technique called *resilient substitution* to scale-out arbitrary topologies. To scale a particular operator, the technique finds the smallest directed acyclic graph (DAG) containing the operator, and stops the inflow of tuples into this DAG. The DAG is then scaled-out by adding more nodes, and the flow is restarted. By providing generic mechanisms for scale-out, these techniques incur high overhead. In contrast, we propose lightweight protocols for load-balancing and route-modifications using global acking framework.

Workflow re-optimization has been proposed for batch-processing-systems such as MapReduce [4]. Their focus is on improving the job workflow based on statistics collected from previous job executions. Examples of workflow changes include: pushing filters upstream, changing the algorithms used for joins based on data-sizes, etc. In contrast, we focus on optimizing group communication topologies and fine-grained assignment of operators to resources.

Proposed schedulers for stream-engines, such as COLA [17], SODA [31], do not consider optimizing complex group communication topologies. Furthermore, their resource-assignment framework is coarse-grained, where an entire operator-replica is considered for assignment. In contrast, by leveraging our *route-maps* representation of the topology, we separately optimize the group-communication operations irrespective of the original stream-topology. Furthermore, by using consistent-hashing for tuple-routing, we propose methods for fine-grained key-space (bucket) assignment.

Improving support for complex analytics applications in databases is an emerging field. Many new system designs have been recently proposed [23, 12]. In contrast, we focus on improving online analytics in the streaming model.

9. CONCLUDING REMARKS

Dynamic compute and network overheads can significantly impact the performance of streaming systems. In this paper, we present efficient techniques for dynamic topology re-optimization, through the use of a feedback-driven control loop. We present a novel technique for network-aware tuple routing using consistent hashing, that improves stream flow throughput in the presence of a number of runtime overheads. We also present methods for dynamic optimization of overlay topologies for group communication operations. To enable fast topology re-optimization with least system-disruption, we present a light-weight, fault-tolerant protocol. All of the proposed techniques are implemented

in a real system and comprehensively validated on three real applications. We demonstrate significant improvement in performance (20% to 200%), while dealing with various compute and network bottlenecks. We show that our performance improvements are robust to dynamic changes, as well as complex congestion patterns. Given the importance of stream processing systems and the ubiquity of dynamic network state in cloud environments, our results represent a significant and practical improvement.

10. REFERENCES

- [1] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. hyon Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryzkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the borealis stream processing engine. In *CIDR*, 2005.
- [2] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *VLDB Journal*, 2003.
- [3] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford. A reliable effective terascale linear learning system. *CoRR*, abs/1110.4198, 2011.
- [4] S. Agarwal, S. Kandula, N. Bruno, M.-C. Wu, I. Stoica, and J. Zhou. Re-optimizing data-parallel computing. *NSDI*, 2012.
- [5] O. Beaumont, L. Marchal, and Y. Robert. Broadcast trees for heterogeneous platforms. *IPDPS*, 2005.
- [6] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. *NIPS*, 2008.
- [7] L. Bottou and Y. LeCun. Large scale online learning. In *NIPS*, 2003.
- [8] R. Castro Fernandez, M. Migliavacca, E. Kalyvianaki, and P. Pietzuch. Integrating scale out and fault tolerance in stream processing using operator state management. *SIGMOD*, 2013.
- [9] B. Chandramouli, J. Goldstein, and S. Duan. Temporal analytics on big data for web advertising. *ICDE*, 2012.
- [10] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal distributed online prediction using mini-batches. *J. Mach. Learn. Res.*, 2012.
- [11] M. A. distributed system kernel. mesos.apache.org.
- [12] X. Feng, A. Kumar, B. Recht, and C. Ré. Towards a unified architecture for in-rdbms analytics. *SIGMOD*, 2012.
- [13] M. D. Hoffman, D. M. Blei, and F. Bach. Online learning for latent dirichlet allocation. In *NIPS*, 2010.
- [14] K. Jacobsson, L. L. H. Andrew, K. H. Johansson, S. H. Low, and et al. Ack-clocking dynamics: Modelling the interaction between windows and the network. *INFOCOMM*, 2008.
- [15] Z. Jerzak and H. Ziekow. The debts 2014 grand challenge. *DEBS*, 2014.
- [16] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. *STOC*, 1997.
- [17] R. Khandekar, K. Hildrum, S. Parekh, D. Rajan, J. Wolf, K.-L. Wu, H. Andrade, and B. Gedik. Cola: Optimizing stream processing applications via graph partitioning. *Middleware*, 2009.
- [18] Y. Kwon, M. Balazinska, and A. Greenberg. Fault-tolerant stream processing using a distributed, replicated file system. *Proc. VLDB Endow.*, Aug. 2008.
- [19] W. Lam, L. Liu, S. Prasad, A. Rajaraman, Z. Vacheri, and A. Doan. Muppet: Mapreduce-style processing of fast data. *Proc. VLDB Endow.*, 2012.
- [20] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying suspicious urls: An application of large-scale online learning. *ICML*, 2009.
- [21] I. A. network bandwidth measurement tool. <https://github.com/esnet/iperf>.
- [22] J. Nishimura and J. Ugander. Restreaming graph partitioning: Simple versatile algorithms for advanced balancing. *SIGKDD*, 2013.
- [23] F. Niu, B. Recht, C. R., and S. J. Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.
- [24] Z. Qian, Y. He, C. Su, Z. Wu, H. Zhu, T. Zhang, L. Zhou, Y. Yu, and Z. Zhang. Timestream: Reliable stream computation in the cloud. *EuroSys*, 2013.
- [25] S. D. realtime computation system. storm.apache.org.
- [26] S. D. stream computing platform. incubator.apache.org/s4/.
- [27] A. S. D. stream processing framework. samza.incubator.apache.org.
- [28] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and evaluation of a real-time url spam filtering service. In *IEEE Symposium on Security and Privacy*, SP, 2011.
- [29] V. Wabbit. <https://github.com/JohnLangford/vowpalwabbit>.
- [30] R. Wagle, H. Andrade, K. Hildrum, C. Venkatramani, and M. Spicer. Distributed middleware reliability and fault tolerance support in system s. *DEBS*, 2011.
- [31] J. Wolf, N. Bansal, K. Hildrum, S. Parekh, D. Rajan, R. Wagle, K.-L. Wu, and L. Fleischer. Soda: An optimizing scheduler for large-scale stream-based distributed computer systems. *Middleware*, 2008.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	14097P
		Application Number	
Title of Invention	Dynamic, Network-Award Topology Optimization for Accelerating Stream Processing Applications		
The application data sheet is part of the provisional or nonprovisional application for which it is being submitted. The following form contains the bibliographic data arranged in a format specified by the United States Patent and Trademark Office as outlined in 37 CFR 1.76. This document may be completed electronically and submitted to the Office in electronic format using the Electronic Filing System (EFS) or the document may be printed and included in a paper filed application.			

Secrecy Order 37 CFR 5.2

☐	Portions or all of the application associated with this Application Data Sheet may fall under a Secrecy Order pursuant to 37 CFR 5.2 (Paper filers only. Applications that fall under Secrecy Order may not be filed electronically.)
--------------------------	---

Inventor Information:

Inventor 1					Remove	
Legal Name						
Prefix	Given Name	Middle Name	Family Name	Suffix		
	Srimat		Chakradhar			
Residence Information (Select One) ☒ US Residency ☐ Non US Residency ☐ Active US Military Service						
City	Manalapan	State/Province	NJ	Country of Residence i	US	
Mailing Address of Inventor:						
Address 1		1 Anne Court				
Address 2						
City	Manalapan	State/Province	NJ			
Postal Code	07726	Country i	US			
Inventor 2					Remove	
Legal Name						
Prefix	Given Name	Middle Name	Family Name	Suffix		
	Naresh		Rapolu			
Residence Information (Select One) ☒ US Residency ☐ Non US Residency ☐ Active US Military Service						
City	Lafayette	State/Province	IN	Country of Residence i	US	
Mailing Address of Inventor:						
Address 1		2501 Soldiers Home Road				
Address 2		Apt. 9D				
City	Lafayette	State/Province	IN			
Postal Code	47906	Country i	US			
All Inventors Must Be Listed - Additional Inventor Information blocks may be generated within this form by selecting the Add button.						
Add						

Correspondence Information:

Enter either Customer Number or complete the Correspondence Information section below. For further information see 37 CFR 1.33(a).
--

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	14097P
		Application Number	
Title of Invention	Dynamic, Network-Award Topology Optimization for Accelerating Stream Processing Applications		

☐ An Address is being provided for the correspondence information of this application.

Customer Number	20844		
Email Address	joseph.kolodka@nec-labs.com	Add Email	Remove Email

Application Information:

Title of the Invention	Dynamic, Network-Award Topology Optimization for Accelerating Stream Processing Applications		
Attorney Docket Number	14097P	Small Entity Status Claimed ☐	
Application Type	Provisional		
Subject Matter	Utility		
Total Number of Drawing Sheets (if any)		Suggested Figure for Publication (if any)	

Filing By Reference :

Only complete this section when filing an application by reference under 35 U.S.C. 111(c) and 37 CFR 1.57(a). Do not complete this section if application papers including a specification and any drawings are being filed. Any domestic benefit or foreign priority information must be provided in the appropriate section(s) below (i.e., "Domestic Benefit/National Stage Information" and "Foreign Priority Information").

For the purposes of a filing date under 37 CFR 1.53(b), the description and any drawings of the present application are replaced by this reference to the previously filed application, subject to conditions and requirements of 37 CFR 1.57(a).

Application number of the previously filed application	Filing date (YYYY-MM-DD)	Intellectual Property Authority or Country

Publication Information:

☐ Request Early Publication (Fee required at time of Request 37 CFR 1.219)

☐ **Request Not to Publish.** I hereby request that the attached application not be published under 35 U.S.C. 122(b) and certify that the invention disclosed in the attached application **has not and will not** be the subject of an application filed in another country, or under a multilateral international agreement, that requires publication at eighteen months after filing.

Representative Information:

Representative information should be provided for all practitioners having a power of attorney in the application. Providing this information in the Application Data Sheet does not constitute a power of attorney in the application (see 37 CFR 1.32). Either enter Customer Number or complete the Representative Name section below. If both sections are completed the customer Number will be used for the Representative Information during processing.

Please Select One:	☒ Customer Number	☐ US Patent Practitioner	☐ Limited Recognition (37 CFR 11.9)
Customer Number	20844		

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	14097P
		Application Number	
Title of Invention	Dynamic, Network-Award Topology Optimization for Accelerating Stream Processing Applications		

Domestic Benefit/National Stage Information:

This section allows for the applicant to either claim benefit under 35 U.S.C. 119(e), 120, 121, or 365(c) or indicate National Stage entry from a PCT application. Providing this information in the application data sheet constitutes the specific reference required by 35 U.S.C. 119(e) or 120, and 37 CFR 1.78.

When referring to the current application, please leave the application number blank.

Prior Application Status		Remove	
Application Number	Continuity Type	Prior Application Number	Filing Date (YYYY-MM-DD)
Additional Domestic Benefit/National Stage Data may be generated within this form by selecting the Add button.			Add

Foreign Priority Information:

This section allows for the applicant to claim priority to a foreign application. Providing this information in the application data sheet constitutes the claim for priority as required by 35 U.S.C. 119(b) and 37 CFR 1.55(d). When priority is claimed to a foreign application that is eligible for retrieval under the priority document exchange program (PDX) the information will be used by the Office to automatically attempt retrieval pursuant to 37 CFR 1.55(h)(1) and (2). Under the PDX program, applicant bears the ultimate responsibility for ensuring that a copy of the foreign application is received by the Office from the participating foreign intellectual property office, or a certified copy of the foreign priority application is filed, within the time period specified in 37 CFR 1.55(g)(1).

Remove			
Application Number	Country ⁱ	Filing Date (YYYY-MM-DD)	Access Code ^j (if applicable)
Additional Foreign Priority Data may be generated within this form by selecting the Add button.			Add

Statement under 37 CFR 1.55 or 1.78 for AIA (First Inventor to File) Transition Applications

☐ This application (1) claims priority to or the benefit of an application filed before March 16, 2013 and (2) also contains, or contained at any time, a claim to a claimed invention that has an effective filing date on or after March 16, 2013. NOTE: By providing this statement under 37 CFR 1.55 or 1.78, this application, with a filing date on or after March 16, 2013, will be examined under the first inventor to file provisions of the AIA.

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	14097P
		Application Number	
Title of Invention	Dynamic, Network-Award Topology Optimization for Accelerating Stream Processing Applications		

Authorization to Permit Access:

☐ Authorization to Permit Access to the Instant Application by the Participating Offices

If checked, the undersigned hereby grants the USPTO authority to provide the European Patent Office (EPO), the Japan Patent Office (JPO), the Korean Intellectual Property Office (KIPO), the World Intellectual Property Office (WIPO), and any other intellectual property offices in which a foreign application claiming priority to the instant patent application is filed access to the instant patent application. See 37 CFR 1.14(c) and (h). This box should not be checked if the applicant does not wish the EPO, JPO, KIPO, WIPO, or other intellectual property office in which a foreign application claiming priority to the instant patent application is filed to have access to the instant patent application.

In accordance with 37 CFR 1.14(h)(3), access will be provided to a copy of the instant patent application with respect to: 1) the instant patent application-as-filed; 2) any foreign application to which the instant patent application claims priority under 35 U.S.C. 119(a)-(d) if a copy of the foreign application that satisfies the certified copy requirement of 37 CFR 1.55 has been filed in the instant patent application; and 3) any U.S. application-as-filed from which benefit is sought in the instant patent application.

In accordance with 37 CFR 1.14(c), access may be provided to information concerning the date of filing this Authorization.

Applicant Information:

Providing assignment information in this section does not substitute for compliance with any requirement of part 3 of Title 37 of CFR to have an assignment recorded by the Office.

Applicant 1

[Remove](#)

If the applicant is the inventor (or the remaining joint inventor or inventors under 37 CFR 1.45), this section should not be completed. The information to be provided in this section is the name and address of the legal representative who is the applicant under 37 CFR 1.43; or the name and address of the assignee, person to whom the inventor is under an obligation to assign the invention, or person who otherwise shows sufficient proprietary interest in the matter who is the applicant under 37 CFR 1.46. If the applicant is an applicant under 37 CFR 1.46 (assignee, person to whom the inventor is obligated to assign, or person who otherwise shows sufficient proprietary interest) together with one or more joint inventors, then the joint inventor or inventors who are also the applicant should be identified in this section.

[Clear](#)

☒ Assignee

☐ Legal Representative under 35 U.S.C. 117

☐ Joint Inventor

☐ Person to whom the inventor is obligated to assign.

☐ Person who shows sufficient proprietary interest

If applicant is the legal representative, indicate the authority to file the patent application, the inventor is:

Name of the Deceased or Legally Incapacitated Inventor :

If the Applicant is an Organization check here. ☒

Organization Name

NEC Laboratories America, Inc.

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	14097P
		Application Number	
Title of Invention	Dynamic, Network-Award Topology Optimization for Accelerating Stream Processing Applications		

Mailing Address Information For Applicant:			
Address 1	4 Independence Way		
Address 2	Suite 200		
City	Princeton	State/Province	NJ
Country i	US	Postal Code	08540
Phone Number		Fax Number	
Email Address			
Additional Applicant Data may be generated within this form by selecting the Add button. Add			

Assignee Information including Non-Applicant Assignee Information:

Providing assignment information in this section does not substitute for compliance with any requirement of part 3 of Title 37 of CFR to have an assignment recorded by the Office.

Assignee 1				
Complete this section if assignee information, including non-applicant assignee information, is desired to be included on the patent application publication. An assignee-applicant identified in the "Applicant Information" section will appear on the patent application publication as an applicant. For an assignee-applicant, complete this section only if identification as an assignee is also desired on the patent application publication.				
Remove				
If the Assignee or Non-Applicant Assignee is an Organization check here. ☐				
Prefix	Given Name	Middle Name	Family Name	Suffix
Mailing Address Information For Assignee including Non-Applicant Assignee:				
Address 1				
Address 2				
City		State/Province		
Country i		Postal Code		
Phone Number		Fax Number		
Email Address				
Additional Assignee or Non-Applicant Assignee Data may be generated within this form by selecting the Add button. Add				

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	14097P
		Application Number	
Title of Invention	Dynamic, Network-Award Topology Optimization for Accelerating Stream Processing Applications		

Signature:

Remove

NOTE: This form must be signed in accordance with 37 CFR 1.33. See 37 CFR 1.4 for signature requirements and certifications.

Signature	/Joseph Kolodka/		Date (YYYY-MM-DD)	2015-03-13	
First Name	Joseph	Last Name	Kolodka	Registration Number	39731

Additional Signature may be generated within this form by selecting the Add button.

Add

This collection of information is required by 37 CFR 1.76. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 23 minutes to complete, including gathering, preparing, and submitting the completed application data sheet form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

Privacy Act Statement

The Privacy Act of 1974 (P.L. 93-579) requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C. 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether the Freedom of Information Act requires disclosure of these records.
2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.
3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.
4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).
5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.
6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).
7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (i.e., GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.
8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspections or an issued patent.
9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.