Detouring and Replication for Fast and Reliable Internet-Scale Stream Processing

Christopher McConnell, Fan Ping, Jeong-Hyon Hwang
Department of Computer Science
University at Albany - State University of New York
{ctm, apping, jhh}@cs.albany.edu

ABSTRACT

iFlow is a replication-based system that can achieve both fast and reliable processing of high volume data streams on the Internet scale. iFlow uses a low degree of replication in conjunction with detouring techniques to overcome network congestion and outages. Computation over iFlow can be expressed as a graph of operators. To cope with varying system conditions these operators continually migrate in a manner that improves performance and availability at the same time.

In this paper, we first provide an overview of our iFlow system. Next, we detail how our detouring technique works in the face of network failures to provide high availability for time critical applications. The paper also includes a description of our implementation and preliminary evaluation results demonstrating that iFlow outperforms previous solutions with less overhead. Finally, the paper concludes with our plans for enhancing replication and detouring capabilities.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, availability, and serviceability; D.2.8 [**Database Management**]: Systems—*Distributed databases*

General Terms

Design, Experimentation, Management, Performance, Reliability

Keywords

availability, fault tolerance, replication, stream processing, detouring

1. INTRODUCTION

Recently, there has been significant interest in applications where high-volume, continuous data streams are persistently generated at diverse geographic locations and distant users must receive the results of processing such data in near real-time. Examples of these applications include online monitoring of Internet components, real-time processing of financial data streams, and sensor-based global environment monitoring.

These Internet-scale stream processing applications can be facilitated by a system that can express the desired computation as a graph of operators [3, 5, 6] and then instantiate such operators over a large number of nodes [18, 21]. Stream processing systems typically provide off-the-shelf operators capable of fundamental processing such as filtering, aggregation and join that are essential to typical data management applications. In addition, custom operators for specific applications can be developed and plugged into the system.

To develop a successful Internet-scale stream processing system, we need to address challenges that arise due to node and link failures. Studies have found that end-to-end communication in the Internet may have, on average, failure rates of at least 0.5%, but no more than 2.4% [7, 15]; an overwhelming percentage of these failures occur at the network (link) level. Further, these outages can take anywhere from seconds to several minutes to be detected and corrected [17]. Although node failures occur less frequently (up to 0.21% of the time [17]), they can cause a loss of essential data or temporary stoppage in data transmission. These failures can adversely affect stream-processing applications. In financial market monitoring, for example, missing trading opportunities due to node and network failures may result in a substantial revenue loss.

The effectiveness of a solution to the aforementioned problems can be measured in terms of three metrics: availability (probability that the end user will receive the results within the predefined time constraint), overhead (cost of additional network and processing resources), and impact on the performance (average latency measured for all users to receive the results).

We observe that previous techniques which aim to handle node and link failures have an important limitation when used for Internet-scale stream processing [4, 10, 11, 12, 20]: they provide only one means of correction, namely replication of operators. The degree of replication is typically determined by the desired level of tolerance for network failures rather than the less frequent node failures. Such replication techniques can achieve improvements in availability

but they incur a large overhead cost.

We present iFlow, an Internet-scale stream processing system that utilizes detouring (discovery of a new route via a remote node) in combination with replication to address the aforementioned limitations. The main goals of iFlow are to (1) improve the system availability while (2) lowering the cost of additional network and processing resources, and finally (3) provide a low average latency measured over all nodes for receiving processed data. iFlow achieves these goals by utilizing detouring, along with operator replication techniques.

A major contributor to Internet-scale stream processing systems' overhead is the creation, migration and removal of operator replicas. iFlow incurs less overhead costs by creating fewer replicas, while having at least the same level of fault tolerance as previous stream processing systems. This improvement can be accomplished by proactively searching for detours from one node to another and utilizing the most effective detours when a communication problem has been found. The effectiveness of such detours will be described in detail in Section 3.2.

In our approach, each node has a detour planner which receives updates via remote nodes about their current route(s) to other remote nodes. The local detour planner is then responsible for determining potential forwarding nodes that will allow data to be received by some destination node. These forwarders are prioritized based upon performance and availability benefits. The details of this technique are discussed further in Section 3.

In this paper, we present the design and implementation of iFlow, as well as an evaluation study conducted by running our iFlow prototype with network latencies emulated based upon network traffic data collected from PlanetLab [19] over 100 nodes across the globe. The evaluation results substantiate the benefit of our technique over previous techniques that rely solely on replication for fault tolerance. This paper also includes our research plans on deploying replicas and adaptively optimizing them in a manner that best takes advantage of detouring.

The remainder of this paper is organized as follows. In Section 2, we describe the iFlow replication model and replica deployment design, followed by a detailed description of our detouring capabilities in Section 3. Next, we detail the implementation of detouring within iFlow in Section 4. We present our evaluation results in Section 5 and future research plans in Section 6. Section 7 summarizes related work and Section 8 concludes this paper.

2. PRIMARY FUNCTIONS OF IFLOW

In this section, we describe the details of iFlow's replication model and replica deployment strategies. First, we present an overview of the iFlow system in Section 2.1. We follow this with the details of the system, first presenting the replication model in Section 2.2 and replica deployment in Section 2.3. Finally, we discuss how iFlow completely recovers from node failures in Section 2.4.

2.1 System Overview

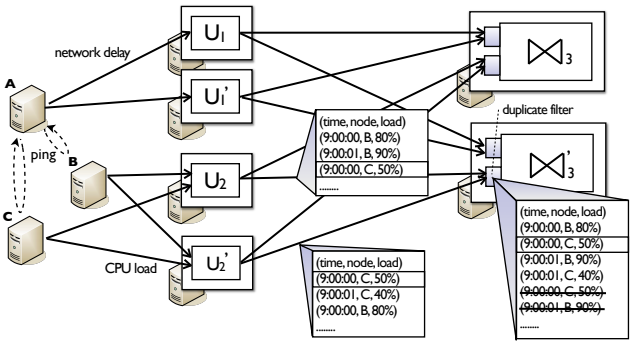

Figure 1: Replication Model. **Nodes B and C report their CPU usage to replicas $\cup_2$ and $\cup'_2$. These replicas merge streams and send the result to $\bowtie'_3$ in parallel. $\bowtie'_3$ uses whichever data arrives first from $\cup_2$ and $\cup'_2$, while ignoring duplicates (see those struck-through).**

iFlow is a distributed software system that aims to facilitate Internet-scale stream processing applications [12]. In iFlow, users express the desired computation as a network of operators. iFlow contains off-the-shelf operators as described previously (Section 1) and allows customized operators to be plugged in as well.

In iFlow, nodes around the globe are partitioned into groups so as to manage the entire system in a scalable fashion. Each of these groups contains tens of nodes at diverse geographic locations with an elected coordinator. This configuration is required so that each group of nodes can efficiently support stream processing that spans distant sources and applications.

2.2 Replication Model

iFlow achieves fast and reliable processing by adopting the replication model presented in our previous work [12]. In this model (illustrated in Figure 1), multiple replicas send data to downstream replicas, allowing the latter to use whichever data arrives first without being hindered by any delayed input flow. To further expedite processing, replicas run without synchronization, possibly processing identical data in a different order as illustrated by replicas $\cup_2$ and $\cup'_2$ in Figure 1. Despite this relaxation, the replication model guarantees that applications always receive the same results as in the non-replicated, fault-free case. This replication model has a distinct advantage of improving performance with more operator replicas. It also allows the system to continue its operation even with the presence of failures which are typically difficult to detect on the Internet scale. For further details about the replication model, we refer the reader to our previous article [12].

2.3 Initial Replica Deployment

Given the replication model described in Section 2.2, a foremost question that arises is the initial deployment of operator replicas. The current iFlow system strives to maximize performance and achieve an adequate level of availability by placing the first instances of operators as in Section 2.3.1 and replicating operators as in Section 2.3.2. Our preliminary designs of a replica deployment technique for further improving availability and a replica migration technique for

coping with changes in system conditions are presented in Sections 6.1 and 6.2, respectively.

2.3.1 Deployment of First Operator Instances

As described in Section 2.1, iFlow forms node groups for scalability reasons. Nodes in the same group periodically obtain the detailed routing and latency information for all communication paths between them. This information is then sent to the elected coordinator.

Whenever the coordinator is requested to instantiate operators, it first selects reliable, under-utilized nodes and then constructs an initial deployment plan where each operator is randomly assigned to one of the selected nodes. The coordinator then refines the plan by repeatedly choosing one operator and reassigning it to a node that would improve performance (i.e., reduce the network delays of the input and output streams of the operator). When the planning reaches an optimized deployment of operators, the coordinator creates the first instances of operators according to the plan.

2.3.2 Deployment of Operator Replicas

In terms of network usage, the initial deployment described in Section 2.3.1 is likely to be sub-optimal, since it cannot consider the data rate of each stream which is unknown when operators are first created. Once first operators are deployed and begin operation, iFlow deploys replicas of operators in a manner that *minimize the overall network cost* similar to operator placement approaches in the non-replication context [1, 18]. The network cost is defined as the sum of individual streams' network costs, each of which is defined as the product of the data rate and the network latency of the stream. This *bandwidth-delay product* is based on the idea that the longer data stays in the network, the more resources it tends to use. An optimal deployment under this metric tends to choose fast network links, thereby ensuring high performance.

Although the above replica deployment strategy achieves low network usage and high performance at the same time, it may place operator replicas to close nodes that have a high risk of being disconnected from the rest of the network at the same time. For this reason, each new operator replica is placed at a node at least a predefined θ distance (e.g., 20 ms in network latency) away from the current operator locations.

The aforementioned replica deployment technique is static and does not take into account detouring capabilities. Our plans to overcome these limitations are described in Sections 6.1 and 6.2.

2.4 Repair

If a node crashes, all the operators at the node are lost. Until new replicas to act on behalf of the lost operators are created, the original fault-tolerance level cannot be guaranteed. To minimize such a period of instability, prompt failure detection and replica reconstruction are required.

Suppose that a node has been sending data to replicas $o_1, o_2, \cdots, o_n$ of an operator. If the node no longer can reach

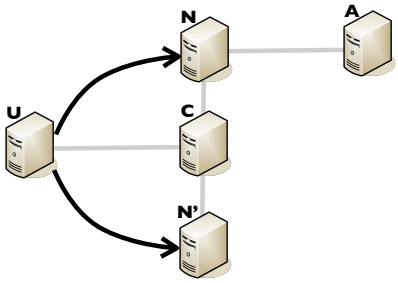

Figure 2: Detouring Example. **Upstream node U may send data to downstream node N through nodes C or N'.**

a subset of operator replicas $\{o_k | i \leq k < j \vee (i \leq k \leq n \wedge 1 \leq k < j)\}$ via all possible detours, iFlow notifies o_j of the unavailability of the above operator replicas. For example, if o_2, o_3 and o_5 among $\{o_1, o_2, o_3, o_4, o_5\}$ are unavailable, o_4 would be informed of the unavailability of o_2 and o_3, and o_1 would be informed of unavailability of o_5. If an operator replica o_j is notified regarding m peer replicas by a majority of the upstream nodes (i.e., the nodes that have been sending data to replicas $\{o_1, o_2, o_3, o_4, o_5\}$), it suspects that these replicas are unavailable due to node failures or isolation from the rest of the network. In this case, m new peer replica of o_j are created and installed at the nodes selected as described in Section 2.3.2.

3. DETOURING

3.1 Detouring Overview

In our approach, each node ensures the delivery of its output based on acknowledgments from the destination node. If a node does not receive an acknowledgment within a predefined time period, it strives to send the data to the destination via other remote nodes. Among such remote nodes, our detouring technique chooses a node that has the best chance of forwarding the data, by selecting fast connections and minimizing the overlap with the current, unavailable route. If the forwarder selected is unsuccessful or too slow we will select the next potential forwarder and attempt to forward the data again. Such a detouring attempt is repeated until the data is actually rerouted to the destination. By utilizing known routes and neighbor nodes, iFlow can overcome network congestion and disconnections with low overhead.

3.2 Detouring

Along with the deployment and use of replicas as described in Section 2.3, iFlow performs detouring as soon as it observes unusual transmission delays to mask network problems, with low overhead. If node U cannot send data to some node N, it checks if any node in the same group can successfully forward data to node N. To carry out detouring efficiently, each node U periodically constructs a detouring plan $\mathfrak{P}[N]$ for each downstream node N. A question that then arises is how iFlow determines what detour plan would be the best option in the face of a communication problem. Given the current output path $\overrightarrow{UN}$, a detour $\overrightarrow{UFN}$ via a remote node F tends to have a higher availability benefit as (1) the overlap between $\overrightarrow{UN}$ and $\overrightarrow{UFN}$ gets smaller and (2) $\overrightarrow{UFN}$ has a smaller delay. For example, detour $\overrightarrow{UAN}$ (precisely, $\overrightarrow{UNAN}$) in Figure 2 has no availability benefit with

regard to $\overrightarrow{UN}$ since $\overrightarrow{UN}$ completely overlaps with $\overrightarrow{UAN}$. Further, $\overrightarrow{UCN}$ is more beneficial than $\overrightarrow{UN'N}$ in that (although $\overrightarrow{UN}$ does not overlap with either $\overrightarrow{UCN}$ or $\overrightarrow{UN'N}$) $\overrightarrow{UCN}$ has a shorter delay, thus can better contribute to performance.

Based on the above observation, we estimate the benefit of each detour and insert K of them to the detouring plan $\mathfrak{P}[N]$ in the decreasing order of benefit. Given the current detouring plan $\mathfrak{P}[N]$, we define the benefit of detour $\overrightarrow{UFN}$ as:

$$\delta(\overrightarrow{UFN}|\mathfrak{P}[N]) := \frac{\text{hops}(\overrightarrow{UN} - \overrightarrow{UFN} - \mathfrak{P}[N])}{\text{hops}(\overrightarrow{UN})} \frac{1}{\text{delay}(\overrightarrow{UFN})} \quad (1)$$

The first term in Definition (1) calculates the proportion (in terms of network hops) of $\overrightarrow{UN}$ that does not overlap with $\overrightarrow{UFN}$ and any detour in the current plan $\mathfrak{P}[N]$. The reason for considering network hops in the definition is that router failures as well as obsolete routing information [2] are substantial contributors to network outages.

If such routing information is not yet available, the detour planner utilizes a pseudo-random forwarder selection procedure. In this case, iFlow selects a forwarder node from within the group randomly, however it utilizes all potential forwarders before re-using any nodes that have already been selected. This ensures that all nodes have an equal chance at forwarding, since no benefit could be calculated.

Given the detouring plan described above, if node U does not receive an acknowledgement from node N within a predefined time threshold (1000 ms in our current implementation), it extracts 2^{i-1} detours from $\mathfrak{P}[N]$ in order at each ith round. To attempt all K detours in $\mathfrak{P}[N]$ within a predefined T seconds, U conducts a detouring round every $\frac{T}{\lceil \log_2 (K+1) \rceil}$ seconds. It is important to note that due to round-trip times a favorable detour may have been successful, but the returning acknowledgement may take longer than the predefined waiting period. To handle this situation, we save the list of forwarders used at the local node to match with the acknowledgement that is received. Upon receiving this acknowledgement, we find the forwarder in the list that was used, mark it as successful and clear the remaining forwarders in the list, thus ignoring any later acknowledgements received.

4. IMPLEMENTATION

For detouring to be successful, iFlow has two main components, the Detour Planner and the Output Manager, which handle the planning and delivery of data streams, respectively. The details of the Benefit Based Detour Planner were discussed in Section 3.2. The Output Manager, which has been implicitly mentioned in Section 2.2, will be detailed in the remainder of this section.

The Output Manager has four primary states, *normal*, *problem detected*, *detour found* and *detouring* (Figure 3). As messages are being passed through the Output Manager to the destination node without any communication problems, the Output Manager is within the *normal* state. When a problem is detected (i.e., the receipt of a message is not

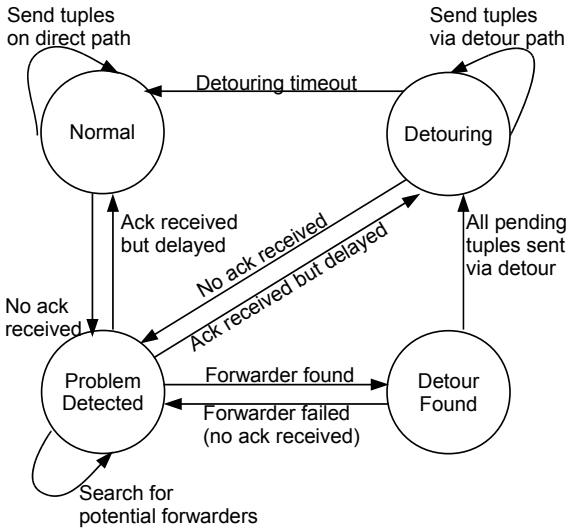

Figure 3: Output Manager State Diagram

acknowledged within a predefined time period), the Output Manager enters the *problem detected* state and begins to take some corrective action. This state is the most complex, as a problem may be detected when the current direct path is slow (but not down), forwarders selected are not working or a forwarder that was working has stopped. If the original path is slow, there is potential that a new forwarder could be found prior to the acknowledgement from the destination. In this case, the Output Manager would enter the *detour found* state. On the other hand, while forwarders are being examined the acknowledgement may be received from the original transmission. In such a scenario, there is no real problem and the Output Manager will return to the *normal* state and continue to use the direct path of communication.

If the problem is a serious one, the Output Manager will continue to ask the Detour Planner for a potential forwarder until one has been found that will successfully forward the messages to the destination. As a forwarder is found, the Output Manager will enter the *detour found* state and attempt to send all buffered messages through this forwarder. If all the messages were successfully sent via the forwarder the Output Manager will enter the *detouring* state, otherwise it will return to the *problem detected* state.

While in the *detouring* state any new messages to be transmitted will utilize the current forwarder specified. If messages are continued to be succesfully forwarded, the Output Manager will remain in this state until the detouring timeout is reached, moving back to the *normal* state and utilizing the direct path again. The *detouring* state is similar to that of the *problem found* state, in that as new messages are passed via the current forwarder there could be abnormal delays, or failures that occur. If the forwarder fails while transmitting new messages the Output Manager will return to the *problem detected* state and begin searching for a new forwarder again. Just as in the *problem detected* state, an abnormal delay may not mean the detour has failed, thus if the acknowledgement is received before a new forwarder is

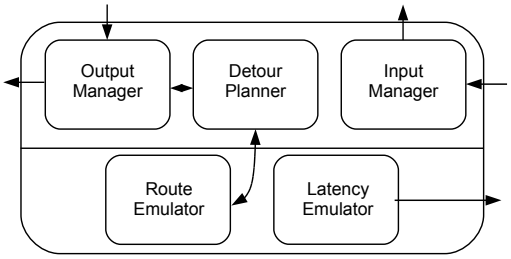

Figure 4: iFlow Node Architecture. **Only the subcomponents related to detouring and their interaction are illustrated.**

found, we simply return to the *detouring* state and continue to send new messages via the detour.

We also provide an overview of the way a node is constructed with regard to detouring in Figure 4. We have limited the components shown to reflect those only required for detouring purposes. The horizontal line separates components used only when network latencies are emulated as described in Section 5 (below the line) and those which are always utilized (above). These emulated latencies are useful for consistent evaluation of algorithms and debugging mechanisms as well. All interactions of the components are shown via the input/output arrows. The output manager and detour planner have been described in some detail, however it is important to note that the output manager not only handles the sending of messages, but the receiving of acknowledgements from other remote node's input manager. The input manager handles the receiving of messages from the sending node's output manager, and also sends out acknowledgement(s) when a message has been successfully received.

The two emulators in Figure 4 can be thought of as an extention of simple file readers. Each emulator will read through specific file types (i.e., route emulator reads the traceroute data files) and parses the recorded data. This data will be described further in Section 5.1. The route emulator recreates the routes from the current node to all remote nodes for use by the detour planner. The latency emulator is not specifically utilized by the node, however it is used to add artificial network delays as in real networking situations and to determine if a node is visible to others (as if a ping request was run).

5. EVALUATION

The current iFlow prototype extends our former stream-processing system [12, 13] with the addition of the detouring capabilities (Section 3). Our iFlow prototype can run on various distributed computing platforms including PlanetLab [19] and server clusters. For evaluation and debugging purposes, iFlow can also be executed while emulating network delays based on network traces.

In this section, we present our preliminary evaluation results obtained by running iFlow in the network emulation mode. We detail our collection of ping and traceroute data used for evaluation in Section 5.1. In Section 5.2 we discuss the settings of our evaluation. Section 5.3 presents the detailed results discussing the impact of replication degree as well as different detouring techniques.

5.1 Data Collection

To compare alternative techniques under identical conditions while ensuring a level of realism, we developed an application to collect traceroute and ping data. This data was obtained over a month on PlanetLab with measurements beginning on September 17, 2009. During this time, 100 nodes from different site locations ran ping measurements (once every minute) and trace route measurements (once every hour) to each of the other nodes. Each node created separate files for pings and traceroutes to each of the other nodes in the measurement group (i.e., each node created 99 ping files, one for each remote node, and 99 traceroute files).

Next, we consolidated each set of data into two files per node for the ping and traceroute measurements respectively. Each line in the files represent the current timestamp the measurement had run, followed by the 100 measurements to each node. One challenge faced was that nodes on PlanetLab are not well managed, as many users can be utilizing nodes causing them to have scheduling conflicts, overloading of the network connections or complete node failure. Even though all measurement applications were launched at the same time there was no guarantee that all nodes were able to perform the measurements at the same time, over the full data collection time frame.

To account for this, we designed a data normalizer, which scanned all the base measurement data, and found the latest start time among all nodes. With a start time defined, we were able to put all timestamps later than this into buckets based upon the interval during which the specified task should have ran. If there was no run for a given bucket (i.e., the node was overloaded/down) we would mark this bucket as incomplete. On the other hand, if a node was able to run it's measurement to a remote node but failed, we marked the bucket as a failure, indicating that the two nodes could not communicate at the given timestamp.

With all data normalized, we found certain nodes to be far too overloaded to give enough information about the interconnectivity with other remote nodes. These nodes were then removed from our set of potential nodes to be utilized during evaluation. This was not a hinderance however, as we were still left with over 90 working nodes, with approximately 4GB of ping and traceroute communication data.

5.2 Settings

The evaluation ran off of the collected trace data as described in Section 5.1. Our evaluation study assumed a system configuration where a large number of nodes around the world are partitioned into groups consisting of 30 nodes each. We then observed the operation of a group while running a network monitoring query. The query involved six nodes that periodically measured the communication latency to each of three remote nodes.

As shown in Figure 1, the latency readings were merged at replicas of Union operator $\cup_1$. To analyze the impact of node load on network delays, the CPU load of the three nodes were first merged at replicas of Union $\cup_2$ and then joined

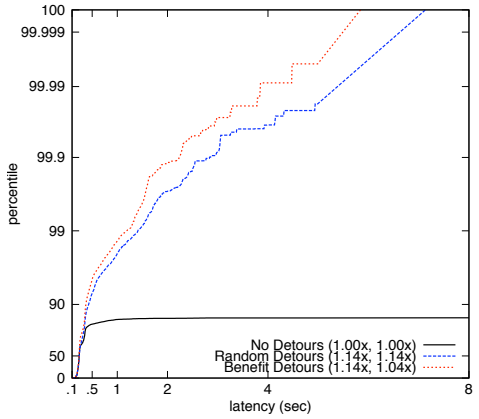

Figure 5: Detouring without Replication. **The effects of different detouring methods are evaluated over 12 hours with a single instance of all operators.**

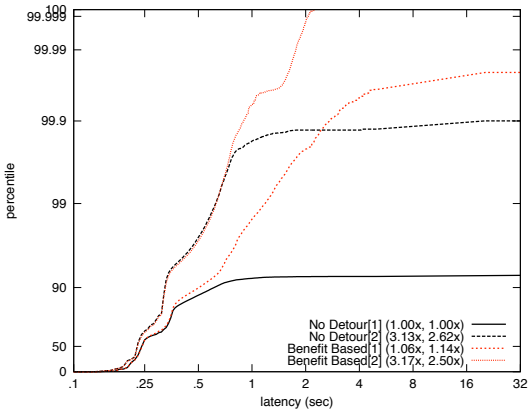

Figure 6: Impact of Detouring on Replication. **The impact of replication and detouring is evaluated over a week time frame. The x-axis is presented on a log scale.**

with latency readings at replicas of Join $\bowtie_3$. This join operation concatenates of the messages from the data streams if they satisfy an equality predicate, such as identical time stamps and node ids. Then, the replicas of $\bowtie_3$ sent their output data (i.e., concatenations of latency and load readings) to Filters that classify the data based on predefined predicates. The outputs of the Filters were then delivered to end applications. All the above replicas were deployed based on our initial deployment algorithm (Section 2.3).

Some specific settings in the evaluation include a maximum detouring timeout of 10 minutes (i.e., after 10 minutes of detouring, iFlow attempts to use the direct path). iFlow utilizes a buffer management system to ensure the transmission of messages, however this buffer cannot hold an infinite number of messages so some must be dropped. While messages are sent from this buffer, periodic routines check the current state of the buffer potentially removing messages. To determine if a message should be dropped, iFlow will compare the current time the buffer examination began with the time that the message was entered into the buffer. If the difference is larger than the message timeout, this message is dropped and will never be delivered to the destination. In our evaluation, this timeout value was set at 8 seconds.

5.3 Results

To evaluate the effects of detouring, we measured the end-to-end latency of output messages as well as the overall availability from the users' perspective. This latency is defined as the difference between the time when an output message was delivered to the destination and the time when the last input data that contributed to the output message was created. The measurement of availability is simply the number of messages received by the destination within an application specific time bound (e.g. 1 second) divided by the total number of messages that the destination would receive in the absence of failures (both network and server). Figures 5 and 6 illustrate the impact detouring has on end-to-end latency (latency), as well as the availability measurement (percentile) of messages received. For example, in Figure 5 focusing on the graph of the No Detouring case, at the point where the latency is 1 second the percentile is only 81.4%. This shows that only 81.4% of the output messages were received by the destination within 1 second of transmission delay. In other words, without replication and detouring, 18.6% of output messages cannot be timely delivered.

Both figures also provide the network bandwidth usage and network cost within the parentheses as a ratio of the base case, No Detours. The network bandwidth usage is determined by the number of messages that are sent over the time evaluation run. As described in Section 2.3.2 the network cost is a product of the data rate of individual streams and the network latency summed across all streams. For example, in Figure 5 the Benefit Detour curve has a bandwidth usage of 1.14 times the No Detouring case over 40 streams. Any detouring will always cause the bandwidth to be larger, however the network cost is dramatically affected by the technique used. By selecting effective forwarders we can keep costs low as seen with a cost of 1.04 in the Benefit Detour graph versus 1.14 in the Random Detour graph.

Figure 5 shows the main benefit that detouring with iFlow can provide over its predecessor [12]. There are no extra replicas of the operators in the query network, however by utilizing detouring (both random and benefit based) techniques the system is able to achieve 100% data transmission versus 17.5% of messages dropped due to the 8 second timeout when no detouring is utilized. Through longer evaluations, we have found that this holds true upwards of 36 hours. We focused on this small 12 hour window to illustrate that failures at the network level can occur early. These early failures can be detrimental to a system that is designed to run for long periods of time and provide high availability (weeks/months even years).

Figure 5 also details the advantages of utilizing the benefit based detour planner over a randomized one. Although both achieve 100% availability, benefit based detouring has a lower network cost (1.04x) compared to the random detouring (1.14x). The reason behind this is that a detour via a remote node may have a shorter network delay compared to

the direct path [2] and our benefit-based detouring technique tends to select such detours keeping delays to a minimum. On the other hand, the network bandwidth is always higher in the detouring case (i.e., 1.14x) as more messages need to be sent as forwarder nodes are utilized. This is the primary cost of utilizing detouring, however our benefit based technique strives to lower the ratio by selecting fewer forwarders (i.e., selecting forwarders that will succeed). Finally, we can see that the maximum end-to-end latency for the benefit based detouring is lower (5.84 seconds) compared to random detouring (7.13 seconds).

Detouring can overcome a lot of network problems, however, it cannot handle the situation where a node with an operator has been completely isolated. Replication will be required to handle this situation, however detouring can provide large benefit when combined with replication. Figure 6 illustrates this point with an emphasis on benefit based detouring and no detouring. The number within the square brackets represents the number of instances of each operator that has been deployed. Thus, the graphs represented with [1] have no additional replicas, whereas graphs with [2] have the original operator(s) with a single replica of each as well, handling a single operator failure without loss.

Even with two instances of each operator, the no detouring case cannot guarantee 100% data transmission, however by utilizing our benefit based detouring, we achieved 100% transmission rate. This figure also implies that there is room to improve the placement of each operator to maximize the benefit of detouring. We can see that even in the single operator case, detouring can significantly increase the successful data transmission, however it cannot reach 100%. With a smarter placement of operators based upon detouring opportunities, this situation might accomplish 100% data transmission. We leave this area of research as future work as described in Section 6.1.

Figure 6 also shows the scalability of our detouring method. We can see that the network cost over a week is still low (1.14x) relative to the non-detouring case for single operator placement. As replicas are added to the system, detouring has the ability to find new routes that are even faster than the direct route taken from a source to destination. This implies that as the number of detours increases the potential to lower the network cost also increases.

6. FUTURE WORK

In this section, we present ways in which we plan to expand iFlow's detouring and replication capabilities. Some of the elements described here have already begun to be implemented.

6.1 Detouring-Aware Replica Deployment

As described in Section 2.3, iFlow's current replica deployment strategy does not consider detouring, thus may provide limited availability guarantees. In this section, we detail detour-aware replica deployment where nodes are selected that can best benefit from new direct and indirect communication opportunities.

Suppose that operator o is already replicated at nodes $N_1, \cdots, N_k$ and we want to deploy a new replica at node N'. In this case, the gain in availability can be defined as the probability that the new replica is reachable (from all upstream/downstream nodes either directly or through detouring) when all the other replicas are not. We are currently developing a technique that can find for each new replica a location that will maximize the gain in availability.

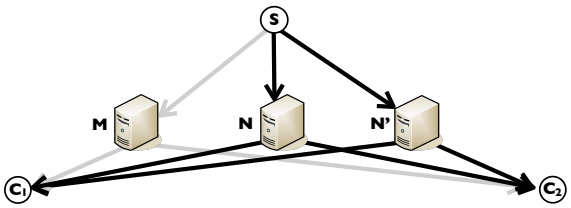

Figure 7: Migration Example. **Operator replicas are running on nodes N and N'. Operator migration from N to M can improve performance from C_1's perspective.**

As an example, assume that an operator at node N in Figure 2 processes data from node U and then consider the problem of replicating the operator at a different node. In this case, node A must not be chosen as a replication point since all possible routes from U to A (e.g., $\overrightarrow{UNA}$, $\overrightarrow{UCNA}$, $\overrightarrow{UN'CNA}$) significantly overlap with routes from U to N, thus the new replica would not receive input data when the operator at N cannot. Conversely, creating a new replica at N' will be beneficial (i.e., improve availability) since N' may receive data from U through $\overrightarrow{UN'}$ or $\overrightarrow{UCN'}$ even if N cannot due to the problem of $\overrightarrow{UN}$ and $\overrightarrow{CN}$.

6.2 Replication-Aware Adaptation

As described in Section 2.3, the initial replica deployment is usually sub-optimal. Further, system conditions tend to change over time. For this reason, we are currently developing an adaptation technique that can periodically migrate operator replicas in a manner that improves both performance and availability.

In our adaptation technique, each node N periodically selects one of its operators (say o) and then, based on the availability benefit as in Section 6.1, selects nodes to which migrating o will improve availability. Among such nodes, N chooses a node to which migrating o will most improve performance. The performance improvement is induced by minimizing the overall network cost as described in Section 2.3 except that the network cost of each data stream in this case is weighted by the stream's impact on clients. The reason behind this cost definition is that the performance (i.e., the end-to-end latency) in general depends on the fastest data flow among multiple replicated flows, thus some streams may have little impact on performance.

As an example, let's assume that nodes N and N' in Figure 7 run operator replicas and then consider the problem of assessing the benefit of migrating the operator replica at N to M. In this case, stream $\overrightarrow{NC_2}$ will be assigned a very low weight because C_2 would usually receive data from $\overrightarrow{N'C_2}$ before it receives the same data from $\overrightarrow{NC_2}$, thus $\overrightarrow{NC_2}$ would have little impact on C_2. On the other hand, both $\overrightarrow{NC_1}$ and

$\overrightarrow{MC_1}$ must be assigned a high weight since C_1 will be affected by them (rather than the slower stream $\overrightarrow{N'C_1}$) before and after migration, respectively. It can be seen that the migration from N to M is advantageous since the end-to-end delay at client C_1 would decrease (while C_2 would not be affected) as C_1 would receive data along a faster route ($\overrightarrow{SMC_1}$) rather than the slower one ($\overrightarrow{SNC_1}$).

When completely implemented, our migration technique will be non-disruptive because it can copy the complete state of each operator without stopping the execution of the operator. For this, we are implementing concurrent copy-on-write data structures [14]. Further, due to other parallel, replicated data flows, any side-effect of operator migration is very likely to be hidden from the end-clients.

6.3 Evaluation of Replica Deployment and Adaptation

Our iFlow prototype is designed to run over PlanetLab and a network emulator that obtains information about network routes and delays from network traffic archives. Emulation-based evaluation provides us the benefit of comparing competing methods under identical conditions, whereas experiments on PlanetLab allows us to obtain real world results. We intend to measure the effectiveness of replica deployment and adaptation techniques by carrying out comprehensive evaluation studies with a variety of stream processing queries and data transmission rates.

7. RELATED WORK

Studies have shown that the majority of end-to-end communication failures are caused by link failures rather than node failures [7, 15]. Many failures were also found to occur in the "middle" nodes between communication endpoints. A common technique to overcome this problem has been the use of overlay networks in which nodes can communicate indirectly via other nodes, possibly routing around outages [2, 16]. The RON (Resilient Overlay Network) system has nodes periodically update their application-level routing tables based on all-pairs ping measurements [2]. Another overlay network called PeerWise focuses on utilizing only a single hop forwarder and limiting the number of required connections for routing around an outage [16]. There have been other approaches that consider detouring at the router level [9, 8]. In all of these previous approaches, network problems last (e.g., 20 seconds on average in RON) until the routing tables become up-to-date through periodic updates. In contrast, iFlow can immediately overcome network problems (usually within a second) by monitoring each message delivery and actively discovering detours when necessary.

Previous techniques for highly-available stream processing typically construct, for each operator, k replicas on independent nodes to mask $(k-1)$ simultaneous node failures. These techniques either execute all the operator replicas [4, 10, 20] or consistently copy the state of a subset of replicas onto other replicas [10, 11, 14]. In contrast to these solutions, our iFlow conducts detouring as soon as it notices a transmission problem. As demonstrated in Section 5, this enables iFlow to achieve higher availability with fewer replicas than previous solutions. iFlow also strives to improve both performance and availability by adaptively constructing and migrating operator replicas.

8. CONCLUSION

This paper discusses achieving fast and highly-available Internet-scale stream processing. In contrast to previous solutions that rely only on replication of operators, our iFlow system combines both replication and detouring (discovery of a new route via a remote node). As shown by our preliminary evaluation, this combination of replication and detouring leads to higher availability with less overhead than previous alternatives. iFlow also manages replicas at diverse geographic locations in a manner that improves both performance and availability.

Our future work will extend the initial work presented in this paper. First, we plan to complete the implementation of iFlow. This implementation effort will include detour-aware replica deployment (Section 6.1) and its incorporation into the repair mechanism (Section 2.4) as well as the replication-aware adaptation technique (Section 6.2). Second, we intend to conduct a comprehensive simulation study with more various stream-processing queries and network trace data. Finally, we plan to obtain real world results by utilizing iFlow and other solutions on PlanetLab.

Acknowledgments

This work is supported in part by the Research and Innovative Technology Administration of the U.S. Department of Transportation through the Region 2 - University Transportation Research Centers (UTRC) Program and the University at Albany through the Faculty Research Awards Program (FRAP) - Category A.

9. REFERENCES

[1] Y. Ahmad and U. Çetintemel. Network-aware Query Processing for Distributed Stream-based Applications. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, pages 456–467, 2004.

[2] D. Anderson, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP)*, pages 131–145, 2001.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 1–16, 2002.

[4] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker. Fault-Tolerance in the Borealis Distributed Stream Processing System. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 13–24, 2005.

[5] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring Streams: A New Class of Data Management Applications. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, pages 215–226, 2002.

[6] S. Chandrasekaran, A. Deshpande, M. Franklin, and J. Hellerstein. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proceedings of*

the 1st Biennial Conference on Innovative Data Systems Research (CIDR), 2003.

[7] M. Dahlin, B. B. V. Chandra, L. Gao, and A. Nayate. End-To-End WAN Service Availability. *IEEE/ACM Transactions on Networking*, 11(2):300–313, 2003.

[8] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the reliability of internet paths with one-hop source routing. In *Proceedings of the 3rd Annual USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 183–198, 2004.

[9] S. W. Ho, T. Haddow, J. Ledlie, M. Draief, and P. Pietzuch. Deconstructing internet paths: An approach for as-level detour route discovery. In *Proceedings of the 8th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2009.

[10] J.-H. Hwang, M. Balazinska, A. Rasin, U. Çetintemel, M. Stonebraker, and S. Zdonik. High-Availability Algorithms for Distributed Stream Processing. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, pages 779–790, 2005.

[11] J.-H. Hwang, U. Çetintemel, and S. Zdonik. A Cooperative, Self-Configuring High-Availability Solution for Stream Processing. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE)*, pages 176–185, 2007.

[12] J.-H. Hwang, U. Çetintemel, and S. Zdonik. Fast and Highly-Available Stream Processing over Wide Area Networks. In *Proceedings of the 24th International Conference on Data Engineering (ICDE)*, pages 804–813, 2008.

[13] J.-H. Hwang, S. Cha, U. Çetintemel, and S. Zdonik. Borealis-R: A Replication-Transparent Stream Processing System for Wide-Area Monitoring Applications. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1303–1306, 2008.

[14] Y. Kwon, M. Balazinska, and A. Greenberg. Fault-tolerant Stream Processing using a Distributed, Replicated File System. In *Proceedings of the 34th International Conference on Very Large Data Bases (VLDB)*, pages 574–585, 2008.

[15] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental study of internet stability and backbone failures. In *Proceedings of the the 29th International Symposium on Fault-Tolerant Computing (FTCS)*, pages 278–285, 1999.

[16] C. Lumezanu, D. Levin, and N. Spring. Peerwise discovery and negotiation of faster paths. In *Proceedings of the 6th Workshop on Hot Topics in Networks (HotNets-VI)*, 2007.

[17] V. Paxon. End-to-End Routing Behavior in the Internet. *IEEE ACM Transactions on Networking*, 5(5):601–615, 1997.

[18] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-Aware Operator Placement for Stream-Processing Systems. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*, pages 49–58, 2006.

[19] PlanetLab. `http://www.planet-lab.org`.

[20] M. A. Shah, J. M. Hellerstein, and E. Brewer. Highly-Available, Fault-Tolerant, Parallel Dataflows. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 827–838, 2004.

[21] Y. Xing, J.-H. Hwang, U. Çetintemel, and S. Zdonik. Providing Resiliency to Load Variations in Distributed Stream Processing. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)*, pages 775–786, 2006.