iFlow: An Approach for Fast and Reliable Internet-Scale Stream Processing Utilizing Detouring and Replication

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

ABSTRACT

We propose to demonstrate iFlow, our replication-based system that supports both fast and reliable processing of data streams over the Internet. iFlow uses a low degree of replication in conjunction with detouring techniques to overcome network outages. iFlow also deploys replicas in a manner that improves performance and availability at the same time, and can cope with varying system conditions by continually migrating replicas.

Based on a live network monitoring application, our demonstration will substantiate the strengths of iFlow. During the demonstration, various visual tools will provide graphical evidence of improvements with regards to availability, performance, and resource usage. To show iFlow's adaptivity, these tools will also allow us to control the demonstration situation including injecting different types of failures.

1. INTRODUCTION

Recent years have seen significant interest in applications where high-volume, continuous data streams are generated at diverse geographic locations and distant users must receive the results of processing the data in near real-time. Examples of these applications include real-time processing of financial data streams, online monitoring of Internet components, and sensor-based monitoring of global environments. These Internet-scale stream processing applications can be facilitated by systems that can express the desired computation as a graph of operators [1, 3, 4] and then instantiate the operators over a large number of nodes [12].

When developing an Internet-scale stream processing system, we need to address the challenges that arise due to node and network failures particularly with end-to-end communications. End-to-end communication over the Internet may have failure rates as high as 2.4% [5, 10]; an overwhelming percentage of these failures occur at the network link level. Detecting and solving such outages may take anywhere from seconds, to several minutes [11], which can be detrimental in a system where response in near-real time is necessary. Although node failures occur less frequently, node failures can cause a loss of essential data or temporary stoppage in data transmission.

We observe that previous techniques that handle node and link failures have an important limitation when used for Internet-scale stream processing [2, 6, 7, 8, 14]: they provide only a single means of correction, namely replication of operators. In such a case, the degree of replication is typically determined by the desired level of tolerance for network failures rather than the less frequent node failures. Previous replication techniques thus incur a large overhead cost to improve availability.

To overcome the aforementioned challenges and limitations, we have implemented iFlow, an Internet-scale stream processing system. The main goals of iFlow are to (1) improve availability (probability of end users receiving the requested data on time) while (2) lowering the cost of additional network and processing resources, and finally (3) improve on the average latency measured over all nodes for receiving the processed data.

To meet these goals, iFlow utilizes a low degree of replication with the following techniques. First, iFlow utilizes detours in the face of network outages. In other words, when a node detects a communication problem, it attempts to send data to the destination node via another node. Our detouring technique proactively identifies such forwarder nodes and prioritizes them based on their likelihood of relaying data under possible network failure scenarios (Section 2.3). Second, iFlow effectively improves availability by replicating operators at nodes that introduce new detouring opportunities (Section 2.2.2). Finally, as network conditions change, iFlow migrates operator replicas in a manner that reduces resource usage and improves both performance and availability (Section 2.4).

In this demonstration proposal, we provide an overview of the iFlow system and then describe the main contributions of our work, focusing on our techniques for replication, detouring, migration and repair. We conclude with our detailed plans for displaying the impact of our contributions.

2. MAIN FEATURES OF IFLOW

iFlow strives to outperform its earlier versions [6, 8] and other previous solutions [2, 14] in performance, availability and resource efficiency through detouring, replica deployment and migration. iFlow groups nodes 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 an elected coordinator of the group. The coordinator is responsible for deploying replicas and migrating them to adaptively improve performance and availability.

2.1 Replication Model

iFlow adopts our previous replication framework [8]. In this framework (illustrated in Figure 1), multiple replicas send data to

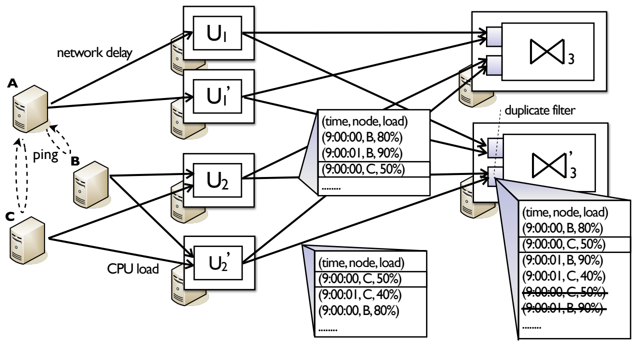


Figure 1: Replication Model. **Nodes B and C report their CPU usage to replicas $\cup_2$ and $\cup'_2$. These replicas merge streams and send the result to $\bowtie'_3$ in parallel. $\bowtie'_3$ uses whichever tuple arrives first from $\cup_2$ and $\cup'_2$, while ignoring duplicates (see those struck-through).**

downstream replicas, allowing the latter to use whichever data arrives first, ignoring delayed data. 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, our previous work 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 [8].

2.2 Initial Replica Deployment

Given the replication model described in Section 2.1, a question that arises is the initial deployment of operator replicas. Our solution to this problem is as follows:

2.2.1 Deployment of First Operator Instances

Whenever the coordinator of a node group is requested to instantiate operators, it first selects reliable, under-utilized nodes and 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 the operator 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.

2.2.2 Detouring-Aware Replica Deployment

After creating the first instances of operators, the coordinator deploys k more replicas for each operator to mask up to k simultaneous node failures. During this phase, it is also beneficial to replicate operators in a manner that improves network availability. Thus, given operator o which has already been replicated at nodes $N_1, \cdots, N_k$, the coordinator estimates the gain in availability of replicating o at node N for all possible $N \notin \{N_1, \cdots, N_k\}$ and then replicates o at a node that maximizes the gain.

Since iFlow strives to mask network failures using detouring, the availability gain of replicating o at node N will increase as more nodes are likely to forward messages from upstream nodes to N and from N to downstream nodes, particularly when all the other replicas are isolated from the network. Thus, the availability gain, denoted as $\Delta_{\mathsf{avail}}(N|N_1, ..., N_k)$, is approximated as:

$$\sum_{\texttt{all node } F \notin \{N_1, \cdots, N_k, N\}} \Delta_{\mathsf{avail}}(N|N_1, ..., N_k; F)$$

where $\Delta_{\mathsf{avail}}(N|N_1, \cdots N_k; F)$ denotes the probability that node N can communicate with a potential forwarder F when other nodes $N_1, \cdots N_k$ cannot. This approximation is made under the assumption that forwarder F is highly likely to reach all the upstream and downstream nodes which are usually independent of $N_1, ..., N_k$. $\Delta_{\mathsf{avail}}(N|N_1, \cdots N_k; F)$ is also approximated as $\frac{|\cap_{i=1}^{k} \overrightarrow{N_i F} - \overrightarrow{NF}|}{|\cap_{i=1}^{k} \overrightarrow{N_i F}|} \cdot \frac{|\cap_{i=1}^{k} \overrightarrow{FN_i} - \overrightarrow{FN}|}{|\cap_{i=1}^{k} \overrightarrow{FN_i}|}$ since (1) simultaneous failure of $\overrightarrow{N_1 F}, \cdots, \overrightarrow{N_k F}$ is typically caused by the failure of their common path ($\cap_{i=1}^{k} \overrightarrow{N_i F}$), (2) $\overrightarrow{NF}$ can have a higher availability benefit as it is less dependent on the common path (i.e., the overlap decreases), and (3) routes $\overrightarrow{NF}$ and $\overrightarrow{FN}$ may not be symmetric [11], thus can have different availability benefits.

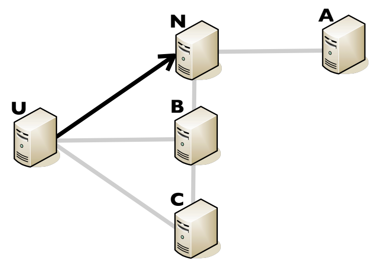


Figure 2: Detouring Example. **Node U sends data to node N. If the link between U and N fails, B and C might be able to forward data from U to N.**

2.3 Detouring

iFlow uses detouring to mask network problems with low replication overhead. If a node U cannot send data to a node N, it checks if any node in the same group can successfully forward data to N. Node U then continues to use the first successful detour until the direct network path to N becomes operational.

To efficiently carry out detouring, each iFlow node U contains a detour planner which receives updates, via remote nodes, about their current route(s) to other remote nodes. Based on this routing information, the detour planner updates, for each remote node N, detour plan $\mathfrak{P}[N]$ consisting of routes that may allow node U to reach N via some remote node in the group. These routes in the detour plan are prioritized based on their latencies and overlap with the direct network path. The reason behind this is that, given the current path $\overrightarrow{UN}$, a detour $\overrightarrow{UFN}$ via a remote node F tends to have a higher benefit as (1) the overlap between $\overrightarrow{UN}$ and $\overrightarrow{UFN}$ gets smaller and (2) $\overrightarrow{UFN}$ has a smaller delay. As described in Section 3, we intend to demonstrate the impact of our benefit-based detouring on both availability and performance.

Figure 2 illustrates an example where node U prioritizes three detours to N (i.e., via A, B and C). In this example, the detour via A (i.e., $\overrightarrow{UNAN}$) has a significant overlap with the original path to N (i.e., $\overrightarrow{UN}$), thus the benefit of the detour is low. On the other hand, both the detours $\overrightarrow{UBN}$ and $\overrightarrow{UCN}$ (i.e., $\overrightarrow{UCBN}$) have high benefits since they do not overlap with $\overrightarrow{UN}$ except the end points. Between them, $\overrightarrow{UBN}$ is a better choice because it involves fewer network hops (i.e., shorter probing time and higher performance) than $\overrightarrow{UCN}$.

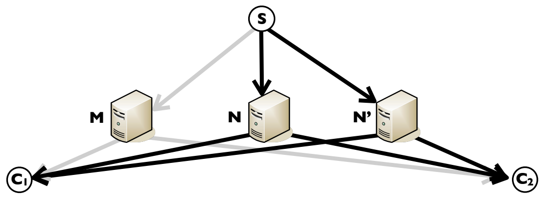


Figure 3: Migration Example. Operator replicas are running on nodes N and N'. Operator migration from N to M can improve performance from C_1's perspective.

2.4 Replication-Aware Adaptation

The system conditions can change over time as the input rate and subsequently the processing load vary, nodes join and leave the system, and regional congestion and failures occur. To adapt to such changes, iFlow continually migrates an operator to a node that further improves performance and availability (estimated as in Section 2.2.2). Each migration copies the state of an operator to a new node without stopping the execution of the operator using a copy-on-write technique [9]. Our migration technique assesses the benefit of migrating an operator based on its impact on end-applications. The reason behind this is that the performance (i.e., the end-to-end latency) in our replication framework depends on the fastest data flow among multiple replicated flows, thus some slow streams may have limited impact on performance.

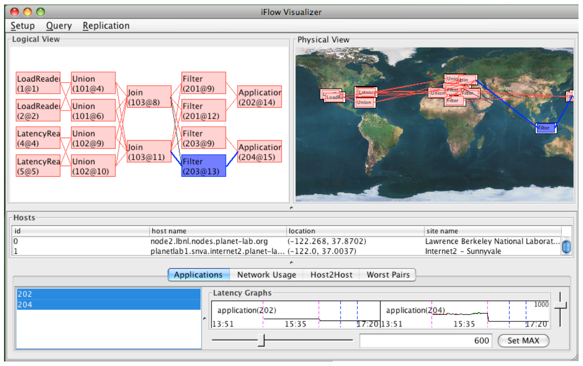
As an example, let's assume that nodes N and N' in Figure 3 run operator replicas and then consider the benefit of migrating the operator replica at N to M. In this case, stream $\overrightarrow{NC_2}$ has a low impact because C_2 would usually receive data from $\overrightarrow{N'C_2}$ before it receives the same data from $\overrightarrow{NC_2}$. On the other hand, both $\overrightarrow{NC_1}$ and $\overrightarrow{MC_1}$ have a high impact 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 remain unaffected) as C_1 would receive data along a faster route ($\overrightarrow{SMC_1}$) rather than the slower one ($\overrightarrow{SNC_1}$).

2.5 Repair

If a node crashes, all the operators of the node become unavailable. In this case, the original fault-tolerance level cannot be guaranteed until new replicas act on behalf of the unavailable operators. To minimize such a period of instability, prompt failure detection and replica reconstruction are required. In our approach, whenever a node cannot send data to a remote operator o via all possible routes, it notifies the coordinator. If the coordinator receives such notifications from a majority of the upstream nodes (i.e., the nodes that send data to o), it suspects that o is lost due to node failure or completely isolated from the rest of the network. In this case, a new replica of o is created by copying an available replica of o as described in Section 2.4. The coordinator uses the replica deployment algorithm in Section 2.2.2 to determine the node to run the new replica.

3. DEMONSTRATION DETAILS

The demonstration will stress the strengths of iFlow using a live network monitoring application. Specifically, we will show that iFlow can efficiently react to changing system conditions by continuously improving both performance and availability as well as effectively masking node and network failures with substantially less overhead than previous solutions.



id	host name	location	site name
0	node2.lbnl.nodes.planet-lab.org	(-122.268, 37.8702)	Lawrence Berkeley National Laborat...
1	planetlab1.snva.internet2.planet-la...	(-122.0, 37.0037)	Internet2 - Sunnyvale

Figure 4: The iFlow Visualizer

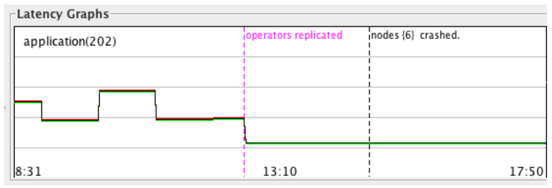


Figure 5: Latency measurements over a sliding window

3.1 Visualizer

The demonstration will proceed using the visualizer shown in Figure 4. The visualizer contains a logical view that displays the queries as networks of operators and a physical view that illustrates the actual deployment of replicas and available nodes around the globe. By utilizing various graphs, the visualizer will display the changes in iFlow measurements across multiple demonstration settings (Section 3.3). The visualizer will also be used to emulate node failures (by killing processes) as well as network congestion and failures (by adding extra delays or terminating connections).

3.2 Setup

The demonstration will show the operation of a node group constructed as described in Section 2. On tens of selected PlanetLab nodes [13] that form a group, we will run a wide area monitoring application. If the demonstration site has a network problem, we will conduct the demonstration only locally while emulating a global networking based on pre-collected network traces. We will demonstrate the impact of detouring by running iFlow with several alternatives (no detouring, random detouring, benefit-based detouring). iFlow also has multiple replica deployment strategies (detour-aware, minimal cost, random), the impact of which can be displayed over multiple demonstrations.

As illustrated in Figures 1 and 4, the demonstration will involve a number of nodes that periodically report CPU load as well as the communication latency for each path to a remote node. The latency readings will be merged at replicas of $\cup_1$ while the load readings will be merged at replicas of $\cup_2$. The replicas of $\bowtie_3$ will associate the CPU load readings and network latency readings based on the Node ID of the CPU readings and the Remote Node ID of the latency readings so that the impact of node overload on network delays can be analyzed later.

3.3 Detouring Impact

iFlow's visualizer contains multiple graphs which show the effectiveness of our proposed detouring techniques in the face of network and node failures. In this section, we describe our demon-

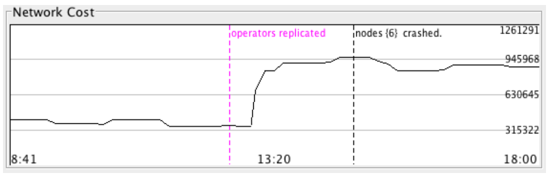


Figure 6: Network cost measurements over a sliding window

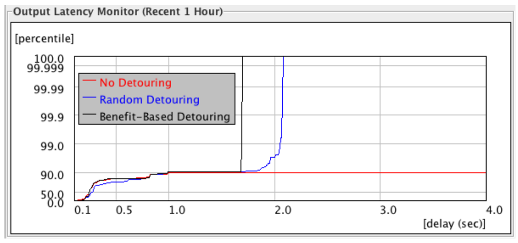


Figure 7: A comparison of detouring techniques in the face of network failures.

stration plan, presenting multiple failure types and a discussion on network cost and bandwidth.

3.3.1 *Node Failures*

In this demonstration, we will cause a node to crash. The visualizer will then show how reactively iFlow can create a new replica to preserve the required fault-tolerance level. This demonstration will verify that our replication approach effectively masks failures as shown in Figure 5 and the repair process is non-disruptive to regular processing (i.e., no significant increase in end-to-end latencies) as described in Section 2.5.

3.3.2 *Network Failures*

Our iFlow visualizer allows us to inject network failures by specifying the path we wish to disconnect, the length of the disconnection period, the frequency of artificial disconnections (i.e., every 5 minutes) and the router to fail. With the router marked, any other path that utilizes this router will also be disconnected.

With failures injected, the visualizer will compare techniques to handle network failures as in Figure 7. To measure the effectiveness of such techniques, we will compare the overall availability of messages sent, as well as the longest latency (in seconds) for all messages to be received by the end user. By changing period and frequency of disconnections we will demonstrate how our detouring technique can address more challenging network instability.

3.3.3 *Network Costs*

We will contrast the network cost of our approach with that of previous approaches which focus on utilizing only replication to handle failures. As partly shown in Figure 6, our iFlow visualizer has graphs depicting both the network bandwidth and the network cost (product of network bandwidth and delay) as measured over the entire system. We will see that iFlow has a lower cost by utilizing a smaller number of replicas compared to previous techniques.

3.4 Adaptivity

This demonstration will begin with creating the first instances of operators and then deploying replicas as described in Section 2.2. During this initial deployment phase, the visualizer will show how replication improves performance (i.e., decreases end-to-end latency) at the expense of increasing network cost (see Figures 5 and 6). We will also demonstrate that our replica migration technique

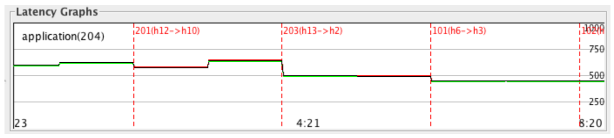


Figure 8: Latency measurements with operator migration

can further improve performance (see Figure 8), availability, and system efficiency as described in Section 2.4. By adding more client applications at new locations, we will show that iFlow can react to such changes by moving operators.

4. REFERENCES

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *PODS*, pages 1–16, 2002.

[2] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker. Fault-Tolerance in the Borealis Distributed Stream Processing System. In *SIGMOD*, pages 13–24, 2005.

[3] 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 *VLDB*, pages 215–226, 2002.

[4] S. Chandrasekaran, A. Deshpande, M. Franklin, and J. Hellerstein. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *CIDR*, 2003.

[5] 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.

[6] J.-H. Hwang, M. Balazinska, A. Rasin, U. Çetintemel, M. Stonebraker, and S. Zdonik. High-Availability Algorithms for Distributed Stream Processing. In *ICDE*, pages 779–790, 2005.

[7] J.-H. Hwang, U. Çetintemel, and S. Zdonik. A Cooperative, Self-Configuring High-Availability Solution for Stream Processing. In *ICDE*, pages 176–185, 2007.

[8] J.-H. Hwang, U. Çetintemel, and S. Zdonik. Fast and Highly-Available Stream Processing over Wide Area Networks. In *ICDE*, pages 804–813, 2008.

[9] Y. Kwon, M. Balazinska, and A. Greenberg. Fault-tolerant Stream Processing using a Distributed, Replicated File System. In *VLDB*, pages 574–585, 2008.

[10] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental study of internet stability and backbone failures. In *FTCS*, pages 278–285, 1999.

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

[12] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-Aware Operator Placement for Stream-Processing Systems. In *ICDE*, pages 49–58, 2006.

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

[14] M. A. Shah, J. M. Hellerstein, and E. Brewer. Highly-Available, Fault-Tolerant, Parallel Dataflows. In *SIGMOD*, pages 827–838, 2004.