A Self-Recovery Technique for Highly-Available Stream Processing over Local Area Networks

Fuyuan Xiao[1,2], Kyoko Nagano[2], Tsuyoshi Itokawa[2], Teruaki Kitasuka[2] and Masayoshi Aritsugi[2]
1) Department of Information Science and Computer Engineering, Guilin University of Technology
12 Jianganlu, Guilin 541004, China
Email: xiaofuyuan@yeah.net
2) Computer Science and Electrical Engineering, Graduate School of Science and Technology, Kumamoto University
2-39-1 Kurokami, Kumamoto 860-8555, Japan
Email: {kyoko@dbms.,itokawa@,kitasuka@,aritsugi@}cs.kumamoto-u.ac.jp

***Abstract*—We present a replication-based and self-recovery-based approach, replica backup, that realizes both continuous and highly-available data stream processing over local area networks. In our approach, we use process-pairs mechanism in which peer operators run in parallel and independently so that each downstream operator can use whichever data arrives first. To further realize continuously stable communication among operators and improve the robustness of system, we devise automatical recovery mechanism that overcomes the limitation of one-off recovery mechanism. In this paper, we first outline the basic design and framework associating with our self-recovery technique. Next, we develop central leader election algorithm (CLEA) that can choose a new operator according with the placement of candidates. This operator placement algorithm that directly measures the latency among operators aims to balance the cost of data stream processing and latency guarantee. Finally, we compare our replica backup method with previous high-available technique through experiments on network simulator ns-3 to demonstrate the utility of our work.**

I. Introduction

Nowadays, there has been an increasing interest in high volume data and continuous data stream with low latency on distributed stream-processing systems. In such systems, data streams are processed in or near real-time for a variety of purposes, such as network monitoring, intrusion detection, real-time analysis and customized e-commerce applications. In those applications domain, continuous and highly-available data stream processing with low latency is critical for dealing with real-world events.

In distributed stream processing systems (DSPSs), the failure of a single server can block data stream processing or cause the loss of a large amount of transient information. DSPSs, therefore, must incorporate a fast and high-availability mechanism with low latency in spite of server failure.

As a response, several data stream processing research prototypes have appeared in [1], [2], [3], [4] which aim at continuous and highly-available dataflow. Because the recovery mechanism results in redirecting input data stream during the failover, they are dissatisfied with the applications that are mentioned above.

To overcome the limitations of previous techniques, we propose an approach using the process-pairs mechanism. In our approach, there is no notion of primary operator or secondary operator over local area networks. Both of peer operators run independently and send in parallel output tuples to each downstream operator so that it can use whichever data arrives first, similar to [5]. Meanwhile, in order to achieve continuous and highly-available data stream processing over local area networks, we also design a system that can automatically choose a new operator to replace the failure one, when an operator fails. The system is then able to tolerate another failure without introducing extra delay.

Previous researches [6], [7], [8], [9] have showed operator placement affects the performance of distributed stream processing applications that could reveal the fundamental trade-off between bandwidth efficiency and result latency. Inspired by [10], for performing node selection based on network location over wide area networks, therefore, we take the placement of operator into consideration in our data stream processing system over local area networks. We will show the details in Section II.

Our solution involves the following subproblems.

*1) **Recovery Mechanism**:* We need a recovery mechanism that realizes continuous and highly-available data stream processing over local area networks.

*2) **Backup Placement Decision**:* We need an algorithm that automatically finds a new appropriate operator after an existing operator is out of work. Our operator placement algorithm strives to balance the cost of data stream processing and latency guarantee.

*3) **Duplicate Tuples**:* In our system, each operator receives input tuples from two upstream operators. For the sake of producing correct results and reducing the operator processing time, it is necessary to eliminate the duplicate tuples from upstream operators.

The remainder of this paper is organized as follows. In Section II, we describe the backup model and how we devise a self-recovery mechanism to achieve continuous and highly-available data stream processing, backup placement decision to balance the cost of data stream processing and latency guarantee, and a duplicate filter to produce correct results and reduce processing time. In Section III, we demonstrate the utility of our work through comparing our approach, replica backup, with a previous method and measuring effect of

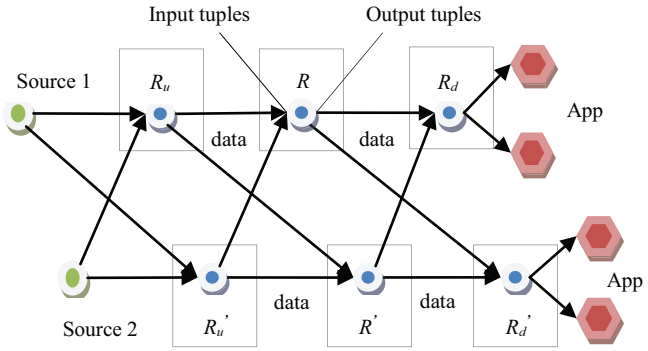

Fig. 1. Example of data stream processing system

selecting *a new replica operator O*
begin
  for *each operator* o_i *which is not connected to stream S* **do**
    $d_{iRu} \leftarrow$ *ping.latency* (o_i, R_u)
    $d_{iR} \leftarrow$ *ping.latency* (o_i, R)
    $d_{iRd} \leftarrow$ *ping.latency* (o_i, R_d)
    average.latency.$d_i \leftarrow (d_{iRu} + d_{iR} + d_{iRd})/3$
  end for
  $O \leftarrow \underset{i}{argmin}\{average.latency.d_i\}$
end

Algorithm 1. Backup placement decision

average server delay, effect of heart-beat interval rate, impact of CLEA and recovery time for random selection and CLEA selection. In Section IV, we present the differences between our approach and previous approaches. In Section V, we summarize the contribution of our research and put forward what we should do in future work.

II. The Backup Model

In this section, we describe the system model how we realize self-recovery, continuous and highly-available data stream processing. We first present the assumptions. Secondly, to improve the robustness of system, we devise a self-recovery mechanism to overcome the limitation of one-off recovery mechanism. Then, we come up with backup placement decision using the central leader election algorithm (CLEA) to balance the cost of data stream processing and latency guarantee. Finally, we describe how we design a duplicate filter for producing correct results and reducing processing time.

A. Assumptions

1) ***System Configuration****:* We assume a small-scale area network or local area network as the substrate for data stream processing. There are 10-20 servers that are grouped into clusters. For simplicity, we assume those servers are filter operators that have been embedded in them, respectively, and have the same and enough capability to process data stream.

2) ***Communication****:* We assume that servers are connected with a fast, reliable, order-preserving and robust message delivery protocol network as TCP.

3) ***Failure Model****:* We assume there is a fail-stop server/network failure. The network failure that isolates server clusters is not supposed to happen in local area networks. We do not consider Byzantine failures [11].

B. A Self-Recovery Mechanism

Generally, rollback recovery mechanism consumes appropriate resources so that it is suitable to be applied for limited-resource environments. However, it introduces extra delay due to redirecting input data stream during the failover [2]. Because the low latency and real-time processing is critical for some kinds of data stream processing applications, it is insufficient for us to use the mechanism to achieve continuous and highly-available data stream processing. Otherwise, the most of research works put much value on one-off recovery so that it could not afford to handle with once more failure. For example, if the primary operator is out of work, the secondary operator will take over it and continuously process the data stream. However, if a failure happens again, namely, the secondary operator is broken, it will stall the data stream processing, and as a result, the downstream operator cannot receive input tuples from upstream. Hence, in our approach, we propose using self-recovery mechanism to overcome the difficulties.

As shown in Figure 1, streams are represented as solid line arrows while operators are represented as circles in the boxes. Operator R_u is said to be upstream of operator R, and operator R_d is said to be downstream of operator R. In this figure, query network is distributed across six nodes, R_u, R, R_d, R'_u, R' and R'_d in which R'_u, R' and R'_d are peers of R_u, R and R_d, respectively. We associate operator R with operator R' and they are in charge of each other. To detect a failure, they periodically send keep-alive requests (heart-beat message) to each other and assume that one of them failed if a few consecutive responses do not return within a timeout period. When one of the operators detects the failure of its peer, the operator that works well will inform its upstream operator to select a new and appropriate operator to replace the failure one with respect to the central leader election algorithm (CLEA) (introduced in the following Subsection) in which the new operator is not part of the data stream processing. For example, as shown in Figure 2, once detecting operator R's failure, operator R' continuously processes input tuples and sends output tuples to each downstream while informing upstream R'_u to choose a new operator instead of R from R_1, R_2, R_3, R_4, R_5 and R_6.

C. Backup Placement Decision

Generally, on the one hand, the distance between two operators is shortest path so that the data stream processing is more faster than others. On the other hand, the distance between peer

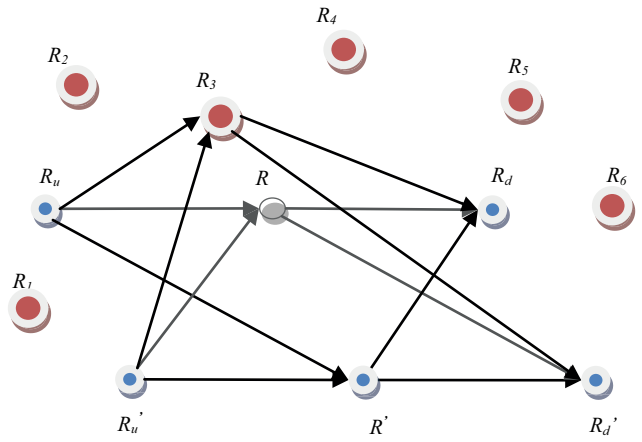

Fig. 2. Example of self-recovery mechanism

operators affects the detection time of operator's failure. To balance the above two aspects, we take the placement of failure one's upstream operator, failure one's peer operator and failure one's downstream operator into consideration. We call such a backup placement decision as central leader election algorithm (CLEA) described in Algorithm 1. Our algorithm is introduced by adapting the discussions in [10] to our target environments. As shown in Figure 2, upstream operator R'_u decides upon the placement of a new operator based on two criteria: one is the operators which have not been used in the stream processing are candidates, such as R_1, R_2, R_3, R_4, R_5 and R_6. The other is these candidates directly measure the latency from them to the targets (failure one's upstream R_u, failure one's peer R' and failure one's downstream R_d), respectively. It aims to balance the cost of data stream processing and latency guarantee.

For instance, when an upstream operator R'_u receives a request to find an operator among the targets R_u, R' and R_d, it informs the free operators R_1, R_2, R_3, R_4, R_5 and R_6 to directly measure the latency $\{d_{1R_u}, d_{1R'}, d_{1R_d}, ..., d_{6R_u}, d_{6R'}, d_{6R_d}\}$ between themselves and the targets, respectively, and compute the average latency $average.latency.d_i = (d_{iR_u} + d_{iR} + d_{iR_d})/3$, respectively. Then these candidates respectively report the average latency $average.latency.d_i$ to upstream operator R'_u for choosing a new operator that has minimal average latency, $argmin\{average.latency.d_i\}$.

D. A Duplicate Filter

In our framework, each operator receives input tuples from two upstream operators. For producing correct results and reducing processing time, we eliminate the duplicate tuples from upstream operators. Our filter algorithm is similar to [5] and is described in Algorithm 2. Because the environments of our approach and [5] are different, we take tuple_id into account while they need to take punctuation and timestamp into consideration.

TABLE I. PARAMETERS AND THEIR DEFAULT VALUES

Parameter	Meaning	Default
λ	input tuple arrival rate (tuples/s)	1000
Tuple	size of a tuple (bytes)	50
Tuple_id	size of a tuple_id (bytes)	8
Network	bandwidth (Mbps)	16
D	average server delay (ms)	10
T	queue-trimming interval (ms)	25
H	heart-beat interval (ms)	100

```
1 eliminating duplicate tuples from upstream operators
2 begin
3     whenever tuple T arrives from upstream operators do
4         if tuple_id of T > max_tuple_id then
5             output(T);
6             max_tuple_id ← tuple_id of T;
7         else
8             discard(T);
9 end
```

Algorithm 2. Duplicate filter

III. Experimental Evaluation

In this section, we present the results that substantiate the utility of our work that consists of some advantage and disadvantage. In Section III-A, we describe how we set up the experiments and simulations. In Section III-B, we compare our technique with upstream backup for continuous and highly-available data stream processing to prove why our approach has good performance. In Sections III-C and III-D, we demonstrate how average server delay and heart-beat interval rate effect interval time/tuple for replica backup and upstream backup during the failover, respectively, which are on benefit of continuous data stream processing. In Section III-E, we compare the network costs of using random algorithm selection and using central leader election algorithm (CLEA). We show CLEA selection balance the cost of data stream processing and latency guarantee. In Section III-F, we also compare CLEA selection with random selection for the recovery time.

A. Setup

The experiments run on a machine with an Intel(R) Core(TM)2 Duo CPU 3.33GHz and 4.00GB main memory underlying Ubuntu 9.10. In our experiments, we created 10 nodes that represent operators in network simulator ns-3 [12], using C++ language. To compare our technique with upstream backup under an identical condition, we used two same input streams that run at 1000 tuples/sec on average. To send tuples and invoke remote procedures, we used TCP

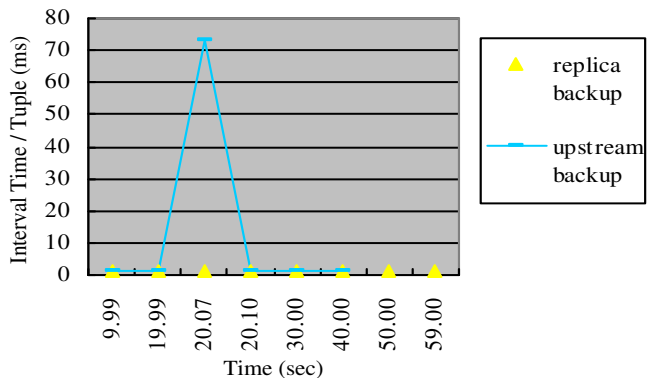

Fig. 3. Comparing for continuous and highly-available stream processing

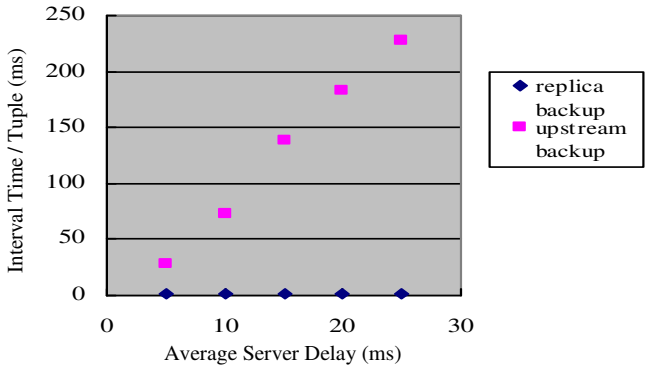

Fig. 4. Interval time/tuple for replica backup and upstream backup as the average server delay varies from 5 ms to 25 ms

sockets. Table I summarizes the main simulation parameters. We set such parameters on the basis of paper [2]. In paper [2], [13] they have proven that the relation between recovery time and bandwidth overhead as the communication interval varies. Therefore, we discuss the tunable parameters: average server delay and heart-beat interval rate and prove how they affect interval time/tuple during the failover in data stream processing.

B. Comparison of Techniques

In this experiment, because the upstream backup method can be implemented most easily among the three methods discussed in [2], we compared our replication-based and self-recovery-based approach, replica backup, with the method for simplicity.

We simulated the experiment in the architecture illustrated in Figure 2 in 1 minute. We set the same server delay, 10 ms, among nodes. We first crashed operator R at 20 sec, then we crashed operator R' at 40 sec. As shown in Figure 3, the triangle labeled "replica backup" shows when sever failures happened in 20 sec or 40 sec there are no effects on data stream processing in which Interval time/tuple keeps about 1 ms. The reason is when operator R was out of work at 20 sec, downstream operators R_d and R'_d could receive the input tuples from operator R' so that it did not introduce extra delay of redirecting input tuples. When operator R' was broken at 40 sec, downstream operators R_d and R'_d could also receive the input tuples from the new operator that had been selected to take the place of the failure operator, R, before.

The line labeled "upstream backup" shows the discontinuous and high delay of redirecting input data stream. In this technique, the primary operator logs their outputs and sends data to downstream, however, the secondary operator keeps idle. Once a primary operator fails, the idle backup takes the place of the failure one by reprocessing the tuples from its upstream's log that is the latest records of failure operator. When we crashed operator R at 20 sec, upstream backup introduced extra delay of redirecting input over 70 ms. Otherwise, after we crashed operator R' at 40 sec, the downstream operator did not produce any output because it no longer received input tuples from upstream operators R' and R that had broken at 20 sec showed in Figure 3.

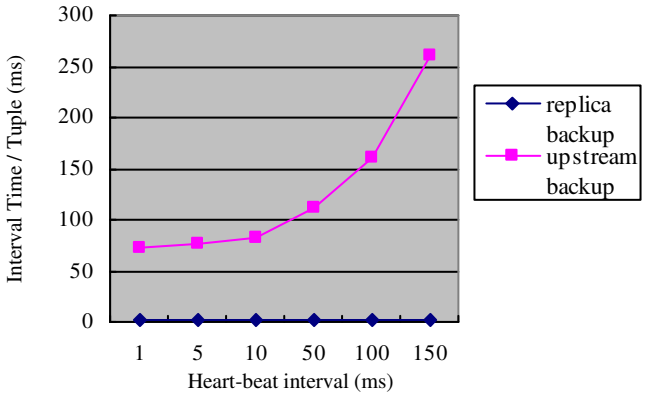

Fig. 5. Interval time/tuple for replica backup and upstream backup as the heart-beat interval varies from 1 ms to 150 ms

Consequently, using our replication-based and self-recovery-based approach, replica backup, in data stream processing system is more stable and robust than using upstream backup.

C. Effect of Average Server Delay

We simulated this experiment in the architecture illustrated in Figure 2 in 1 minute. We set the same server delay among nodes, i.e. 5 ms, 10 ms, 15 ms, 20 ms, and 25 ms, respectively. We only crashed the operator R at 30 sec. Figure 4 shows the relation between interval time/tuple and average server delay during the failover. As the average server delay varies from 5, to 10, 15, 20, and 25 ms, replica backup is the clear winner with stable interval time/tuple about 1 ms because each downstream operator could receive input tuples from operator R'. However, due to redirecting input tuples time, interval time/tuple of upstream backup was in proportion to the average server distance during the failover.

D. Effect of Heart-Beat Interval Rate

In this experiment, we wanted to compare the effect of heart-beat interval for interval time/tuple during the failover.

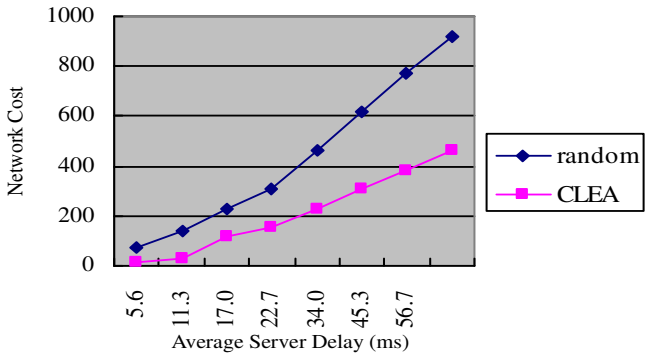

Fig. 6. Network cost for random and CLEA selection as the average server delay varies from 5.6 ms to 56.7 ms

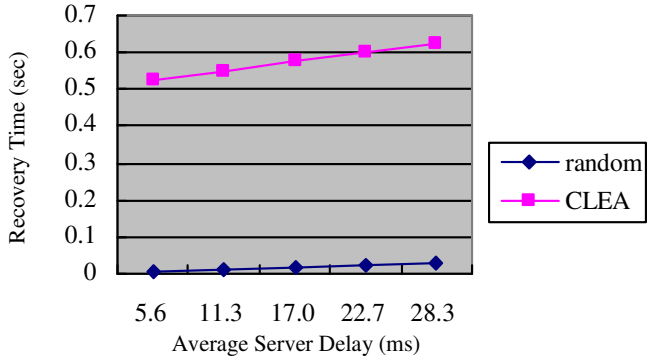

Fig. 7. Recovery time for CLEA and random selection as the average server delay varies from 5.6 ms to 28.3 ms

We also simulated the experiment in the architecture illustrated in Figure 2 in 1 minute and crashed the operator R at 30 sec. We set the same server delay, 10 ms, among nodes. As illustrated in Figure 5, as the heart-beat interval varies from 1, to 5, 10, 50, 100, and 150 ms, replica backup has a better performance than upstream backup because the peer operators run independently and in parallel, and as a result, there were no effects on data stream processing. In contrast, because of redirecting input data stream and resending the tuples from its upstream's log during the failover, upstream backup introduced extra delay. Interval time/tuple of upstream backup was in proportion to the average server delay during the failover.

E. Impact of CLEA

As described in Section II-C, we used the central leader election algorithm (CLEA) in backup placement decision that chooses a new operator to take over the failure one aiming to balance the cost of data stream processing and latency guarantee. In this experiment, we used the architecture illustrated in Figure 2 in 1 minute and crashed operator R at 30 sec. To select a new node that has minimal average latency, we set the different server delay among candidates.

In our framework, we use the following formula to calculate network cost that is the same with [5], $networkcost(S) = rate(S) * latency(S)$ in which $rate(S)$ denotes the data rate of stream S and $latency(S)$ donates the network latency of stream S. We got latencies through $ping$ among operators. We tested how much network of labeled "random" and "CLEA" selection cost as the average server delay varies from 5.6, to 11.3, 17.0, 22.7, 34.0, 45.3, and 56.7 ms, respectively. As shown in Figure 6, randomly choosing a new operator consumed more network cost than using CLEA, and the network cost of using random selection was also increasing faster than using CLEA, because CLEA selection always chose the shortest path to transmit data. Using CLEA in backup placement decision, we can balance the cost of data stream processing and latency guarantee.

F. Comparison of Recovery Time

Although our proposal, replica backup, has many advantages that have been mentioned above, there is also a disadvantage. We simulated the experiment in the architecture illustrated in Figure 2 in 1 minute. To select a new node that has minimal average latency, we also set the different server delay among candidates. We crashed operator R at 30 sec. The recovery time was defined as from upstream operator R'_u received the backup notify to a new operator received the input tuples from upstream operator.

In Figure 7, the square labeled "CLEA" and rhombus labeled "random" selection respectively represent how the recovery time varied, as the average server delay changed from 5.6, to 11.3, 17, 22.7, and 28.3 ms. Because of choosing time (Lines 3-9 in Algorithm 1), CLEA selection has a higher recovery time than random selection over 0.4 sec.

IV. Related Work

Providing continuous and highly-available in data stream processing has been widely researched in distributed systems. There are two general classifications of failover model. One is rollback recovery mechanism. The other is state-machine recovery mechanism. Conventional rollback recovery protocols are categorized into passive standby, upstream backup and active standby. The recovery mechanisms were used in [2], [4] in which they all employ the process-pairs approach of replicated computation in the notion of a primary and secondary. In those techniques, a primary operator periodically sends heart-beat message to the secondary operator to detect the server failure. If the primary operator is out of work, the secondary operator will take over the operation of the failed one. Thus, such implementation introduces extra delay of redirecting input data stream.

Approaches in [1], [3] apply state-machine recovery mechanism in which all the replicas run in parallel but receive data from only one of the upstream. All the replicas also are "up-to-date". Thus, server failure stalls the processing only during the failover.

The previous paper [5] introduces multiple replication-based method in data stream processing over wide area networks

in which there have rich network resources and partitions are more frequent. Their approach also collects garbage and revives replicas to cope with the dynamics of the environment. If failures or local congestions occur, however, the surviving replicas can experience unexpected delays. For solving this problem, they use reviving garbage-collected replicas. Because multiple replicas participate in data flow and garbage-collecting/reviving replicas are used in data stream processing, this method uses more resources than our approach and is unsuitable to be applied in limited-environments.

To the best of our knowledge, conventional rollback recovery mechanism and state-machine recovery mechanism introduce extra delay during the failover and just can support one-off failure of system. In our approach, the peer operators run in parallel and independently. If one server failed, the peer operator can receive input tuples and send output tuples to downstream operator while notifying its upstream to select a new operator to replace the failure one. Thus, the data stream processing is continuous. In addition, the system can support once more server failure after automatically recovery. Hence, our self-recovery solution improves continuous and high-availability for data stream processing over local area networks.

V. Conclusion

In this paper, we argued that the distributed and data flow nature of continuous and high-available data stream processing applications raises novel challenges and opportunities over local area networks. We introduced a replication-based and self-recovery-based approach, replica backup, that can deal with both high interval time/tuple during the failover and continuous fail-stop failures. The central notion behind the technique is to replicate operator and let them work in parallel and independently while automatically recovering server failure. In this way, each downstream operator in the system can use whichever data arrives first from upstream operators to improve the continuous of dataflow. It also can improve the robustness of system through automatically recovering. The system, therefore, naturally achieves the continuous and highly-available data stream processing. During the course of data stream processing, we also devised a duplicate filter for producing correct results and reducing processing time.

Based on this replication and self-recovery framework, we also took the placement of operator into consideration. We introduced the central leader election algorithm (CLEA) to choose a new operator replacing the failure one. We used CLEA to balance the cost of data stream processing and latency guarantee.

Through simulations, we demonstrated the approach that we introduced, replica backup, is better than upstream backup for continuous and highly-available data stream processing. We investigated how the average server delay and heart-beat interval effect the interval time and which method has more stable performance. Finally, we tested how much network cost and showed randomly choosing a new operator has lower performance in terms of network cost and average server delay but has lower recovery time than using CLEA selection.

We currently have a basic framework that can provide continuous and highly-available data stream processing over local area networks. In future, we will extend our system and take network load into consideration. We plan to study the interaction between high availability and load balancing and detect how it effects the data stream processing.

References

[1] M. A. Shah, J. M. Hellerstein, and E. Brewer, "Highly-available, fault-tolerant, parallel dataflows," in *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, June 2004, pp. 827–838.

[2] J.-H. Hwang, M. Balazinska, A. Rasin, U. Çetintemel, M. Stonebraker, and S. B. Zdonik, "High-availability algorithms for distributed stream processing," in *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, April 2005, pp. 779–790.

[3] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker, "Fault-tolerance in the borealis distributed stream processing system," in *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, June 2005, pp. 13–24.

[4] J.-H. Hwang, Y. Xing, U. Çetintemel, and S. B. Zdonik, "A cooperative, self-configuring high-availability solution for stream processing," in *ICDE '07: Proceedings of the 23rd International Conference on Data Engineering*, April 2007, pp. 176–185.

[5] J.-H. Hwang, U. Çetintemel, and S. Zdonik, "Fast and highly-available stream processing over wide area networks," in *ICDE '08: Proceedings of the 24th International Conference on Data Engineering*, April 2008, pp. 804–813.

[6] T. Repantis and V. Kalogeraki, "Replica placement for high availability in distributed stream processing systems," in *DEBS '08: Proceedings of the 2nd International Conference on Distributed Event-Based Systems*, July 2008, pp. 181–192.

[7] Y. Ahmad and U. Çetintemel, "Network-aware query processing for stream-based applications," in *VLDB '04: Proceedings of the 30th International Conference on Very Large Data Bases*, August 2004, pp. 456–467.

[8] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-aware operator placement for stream-processing systems," in *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, April 2006, p. 49.

[9] J. Ledlie, P. Gardner, and M. Seltzer, "Network coordinates in the wild," in *NSDI '07: Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation*, April 2007, pp. 299–311.

[10] B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: a lightweight network location service without virtual coordinates," in *SIGCOMM '05: Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, August 2005, pp. 85–96.

[11] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *OSDI '99: Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, February 1999, pp. 173–186.

[12] The ns-3 network simulator, http://www.nsnam.org/.

[13] K. Nagano, T. Itokawa, T. Kitasuka, and M. Aritsugi, "Exploitation of backup nodes for reducing recovery cost in high availability stream processing systems," in *IDEAS '10: Proceedings of the 14th International Database Engineering and Applications Symposium*, August 2010, pp. 61–63.