Improving Performance of Streaming Applications with Filtering and Control Messages

Peng Li Jeremy Buhler
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO 63130
{pengli, jbuhler}@wustl.edu

ABSTRACT

In streaming computing applications, some data can be filtered to reduce computation and communication. Due to filtering, however, some necessary information might be lost. To recover lost information, we use *control messages*, which carry control information rather than input data. The order between control messages and input data must be *precise* to guarantee correct computations. In this paper, we study the use of control message in suppressing data communication, which improves throughput. To ensure precise synchronization between control messages and input data, we propose a credit-based protocol and prove its correctness and safety. Results show that with the help of control messages, the application throughput can be improved in proportion to filtering ratios.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

Keywords

Streaming Computing, Control Message, Data Throughput

1. INTRODUCTION

Streaming computing is a paradigm of parallel and distributed computing [1, 2]. A typical streaming computing system is a network of computing nodes connected by first-in first-out (FIFO) data channels. Figure 1 shows a streaming system constructed according to Equation 1 to compute variances on *continuous datasets*. The source node u duplicates input data and sends them to v and w, which compute $\overline{z}^2$ and $\overline{z^2}$, respectively. After finishing processing one dataset, $\overline{z}^2$ and $\overline{z^2}$ are then sent to node x to compute variance.

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} z_i^2 - (\frac{1}{n}\sum_{i=1}^{n} z_i)^2 \qquad (1)$$

The pipeline shown in Figure 1 is part of a larger application, VERITAS [3], where a large portion of data are zeroes. To improve throughput, u may filter those zeroes to reduce computations and save communication bandwidth. However, with filtering, the number of tokens received by v and w varies from dataset to dataset. How do v and w know dataset boundaries?

PACT'14, August 24–27, 2014, Edmonton, AB, Canada.
ACM 978-1-4503-2809-8/14/08.
http://dx.doi.org/10.1145/2628071.2671421.

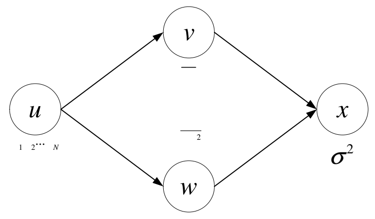

Figure 1: Streaming-computing variance.

2. CONTROL MESSAGES

To indicate dataset boundaries, we turn to *control messages*, which deliver necessary control information (not just dataset boundary) that cannot be represented by data tokens. In the variance application, node u can send control messages to v and w to deliver data boundary information. The order in which control and data are processed should be *precise*: if a node sends the ith data token, followed immediately by a control message, then this message should be processed by the receiving node after computing on all input data but before consuming the ith data token.

There are two ways to deliver control messages. We can embed control messages in data streams and tag each data token and control message to multiplex/demultiplex them. Alternatively, we can create a separate control channel q' for *each* data channel q connecting two nodes. While the tagging method trivially guarantees precise message timing, the separate-channel method needs carefully designed protocols to precisely synchronize control messages with data tokens.

3. A CREDIT-BASED PROTOCOL

In the separate-channel approach, all control messages are delivered through message channels. We use a special message called *credit message* to tell the receivers when to listen at message channels. A credit message carries *credit* to allow the receiver to consume consecutive data tokens without checking for the next control event. The sender and receiver each maintains internal credit balances, which are integer values that are initially zero. When a receiver receives some number c of credits on an edge e, its credit balance RCB_e is incremented by c; when it *consumes* a data token on e, RCB_e is decremented by one. The sender's credit balance

SCB_e is incremented by one whenever it *sends* a data token; when it sends c credits to the receiver on e, *SCB_e* is decremented by c.

Algorithm 1 and Algorithm 2 describe the receiver's protocol and the sender's protocol, respectively. The sender's protocol is parametrized by a threshold T, which should be set less than the buffer size of the outgoing data channel to avoid deadlocks.

Algorithm 1: Receiver Credit Balance Protocol

```
while $RCB = 0$ do
    wait for a control message on $q'$
    let $c$ be credit value carried by message
    if $c = 0$ then
        consume message
    else
        Detach $c$ credits from message
        $RCB \leftarrow RCB + c$
wait for a data token on $q$
consume token
$RCB \leftarrow RCB - 1$
```

Algorithm 2: Sender Credit Balance Protocol

```
if token is ready then
    emit token on $q$
    $SCB \leftarrow SCB + 1$
while control message is ready OR $SCB > T$ do
    emit message on $q'$ with $SCB$ credits
    $SCB \leftarrow 0$
```

We argue that the sender and receiver protocols ensure precise ordering of control messages vs. data tokens.

THEOREM 1. *If a receiver and sender are connected by an edge and behave as in Algorithms 1 and 2, and the sender issues a data token d followed by a control message m, then the receiver will process m after d but before the next token following d.*

PROOF. The sender's protocol never sends the credit necessary to consume a data token before sending the token itself. Hence, when d is sent, the receiver does not have the credit needed to accept it. This credit is sent only with control message m and is sufficient only to process d and any unreceived data tokens sent prior to d. Hence, the receiver sees m, uses its credits to accept d and any prior tokens, and then processes m.

For any data token d' sent after d, the receiver will not receive the credit needed to accept it until after processing m. □

Now we prove that the credit protocol is freedom of deadlocks.

THEOREM 2. *If a receiver and a sender are connected by an edge and behave as in Algorithms 1 and 2, they will never deadlock.*

PROOF. To verify freedom from deadlock, we must check that two bad cases never occur.

- *Case 1.* Sender is blocked writing a full data channel q while receiver is blocked reading an empty control channel q'.
- *Case 2.* Sender is blocked writing a full control channel q' while receiver is blocked reading an empty data channel q.

For Case 1, if the data channel is full, but the receiver is reading the control channel, then the receiver has no credits to consume data tokens. If no control message with credits is in flight, then the sender has sent $|q|$ data tokens without sending any credits. Since the sender's threshold $T = |q|$, it would have sent credits before trying to send token $|q| + 1$, which contradicts the assumption that no credits are in flight. Hence, the receiver will be able to drain the data channel later, and the nodes are not deadlocked.

We now consider Case 2. If the control channel is full, but the receiver is blocked reading the data channel, then the receiver has at least one unexpended credit. But the sender never issues such credits before issuing the corresponding data tokens. Hence, there must be enough data tokens in flight to expend the receiver's credits, and it will consume them and switch to reading the control channel. □

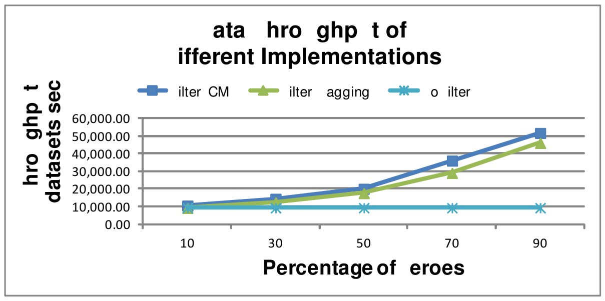

Figure 2: Data throughput (buffer size = 64)

4. EXPERIMENTAL RESULT

We implemented the variance application and measured throughput. Node u generated continuous dataset of 1024 numbers with configured percentages of zero values. We implemented the tagging method (Filter_Tagging) and the separate-channel method (Filter_CM) for filtering, as well as a non-filtering baseline (NoFilter) for comparison. Each node was mapped onto a separate physical processor core and channels were implemented with shared memory. Results show that both filtering methods improve throughput in proportion to the percentage of zero value. The throughputs of the separate-channel method are generally 10% to 20% better than those of the tagging method.

5. CONCLUSION AND FUTURE WORK

In this paper, we demonstrated that the performance of streaming applications can be improved by filtering unnecessary data. To support filtering, control messages might be required to compensate lost information. We proposed a credit-based protocol for synchronizing data tokens with control messages and proved its correctness and safety. With the protocol, the separate-channel method was 10% to 20% better than the tagging method in term of data throughput in our experiment. The application performance is focused on throughput in this paper. In future, we plan to investigate the impact of data filtering on data latency.

6. REFERENCES

[1] P. Li, K. Agrawal, J. Buhler, and R. D. Chamberlain. Deadlock avoidance for streaming computations with filtering. In *ACM Symp. on Parallelism in Algorithms and Architectures*, 2010.

[2] R. Stephens. A survey of stream processing. *Acta Informatica*, 34(7):491–541, 1997.

[3] E. J. Tyson, J. Buckley, M. A. Franklin, and R. D. Chamberlain. Acceleration of atmospheric Cherenkov telescope signal processing to real-time speed with the Auto-Pipe design system. *Nuclear Instruments and Methods*

in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 595(2):474–479, 2008.