Practical Target-Based Synchronization Strategies for Immutable Time-Series Data Tables

Bennett Meares
School of Computing
Clemson University, USA
bmeares@clemson.edu

Amy Apon
School of Computing
Clemson University, USA
aapon@clemson.edu

Mitch Shue
School of Computing
Clemson University, USA
mshue@clemson.edu

Abstract

Extracting data from production databases demands consideration of the stress imposed on the machines, so strategies are required to reduce the transfer of duplicate data. This paper examines immutable, target-based, time-series synchronization strategies to answer the following question: given several scenarios, which strategies best optimize run-time, bandwidth, and accuracy?

Keywords: time-series, database synchronization

1 Introduction

Many applications generate immutable time-series data streams. A persistent problem is regular target-based synchronization, constrained by limited knowledge of and access to the source database (Ahluwalia et al., 2010). Through the constraints of immutability and a datetime axis, data tables may be efficiently updated by several target-based synchronization strategies.

2 Methods

To determine practical strategies, we first choose common immutable time-series scenarios and propose three classes of target-based synchronization strategies. We then evaluate the performances and rank the strategies according to the reader's priorities[1].

2.1 Scenarios

The following immutable time-series data stream scenarios are simulated to increase a source table by one record per ID per hour for one year.

- *single-append-only*
  A single ID makes up the data stream, and backlogging in the source table is not a concern.
- *multiple-large-n-append-only*
  Many IDs (100) make up the data stream, and backlogging in the source table is not a concern.
- *single-known-backlog*
  A single ID makes up the data stream, and records are backlogged into the source table within a known interval.
- *unknown-backlog*
  An unknown number of IDs (3) makes up the data stream, and records are backlogged into the source table within a random interval.

2.2 Strategies

To address the specified scenarios, the strategies are grouped into three classes: (1) **simple syncs**, (2) **iterative syncs**, and (3) **corrective syncs**.

2.2.1 Speed-First: *Simple Syncs*

The *Simple Syncs* class prioritizes run-time and bandwidth over accuracy by focusing on "new" rows.

- *Simple Sync*
  Select rows newer than the latest target datetime.
- *Simple Backtrack Sync*
  Select rows newer than a "walked back" latest target datetime.
- *Simple Slow-ID Sync*
  Select rows newer than the oldest of each ID's latest datetimes.
- *Simple Append Sync*
  Generate *Simple Sync* queries for each ID and append them into a single transaction.
- *Simple Join Sync*
  Left join a temporary table of latest datetimes to emulate *Simple Sync* per each ID.

2.2.2 No-Compromises Accuracy: *Iterative Syncs*

Strategies within *Iterative Syncs* appeal to users for whom accuracy is a non-negotiable requirement. Each strategy may be bounded to decouple the run-time from the underlying tables' sizes.

- *Iterative Simple Sync*
  For each partition of the datetime axis, compare row-counts and perform *Simple Sync* when row-counts differ.
- *Daily Row-Count Sync*
  Build a table of days' row-counts and perform *Simple Sync* on days with differing row-counts.
- *Binary Search Sync*
  For each partition of the datetime axis, compare row-counts and recursively binary search partitions with different row-counts until sufficiently small intervals are encountered. Perform *Simple Sync* on the small intervals.

[1] https://github.com/bmeares/syncx

- *Iterative CPISync*
  For each partition of the datetime axis, compare row-counts and perform *CPISync* (Trachtenberg et al., 2002) when row-counts differ.

2.2.3 Best of Both Worlds: *Corrective Syncs*

Corrective Syncs seek to balance run-time, bandwidth, and accuracy by regularly performing *Simple Sync* and intermittently performing expensive synchronizations to "correct" the target table. The tested corrective strategies are named *Simple Monthly* `<strategy>` *Sync*, where `<strategy>` is an *Iterative Sync* strategy (e.g. *Simple Monthly Iterative CPISync*).

3 Results

Run-time is the cumulative duration in seconds, **bandwidth** is the total number of rows fetched from the source database, and **accuracy** is the ratio of the sizes of the target and source tables. Run-time and bandwidth are normalized into *Choice Indices*, a scale where > 1.0 beats *Simple Sync* and < 0.0 exceeds fifteen times *Simple Sync*'s requirements[2] (see Figure 1).

3.1 Comparing *Simple Syncs*

At a steep expense to run-time, *Join* and *Append* slightly improve bandwidth (at an observed $10:1$ ratio for 100 IDs). *Simple Backtrack Sync*, however, sacrifices bandwidth for accuracy and run-time; a 24-hour backtrack interval doubles the daily bandwidth of *Simple Sync* while increasing its accuracy from ~90% to ~93% for *unknown-backlog* and to ~100% for *single-known-backlog*[3].

3.2 Comparing *Iterative Syncs*

Iterative CPISync maintains 100% accuracy with near-optimal bandwidth performance at a steep cost to run-time ($+\sim 2,800\%$), which is expected given its high execution cost (Tang et al., 2010). *Daily Row-Count Sync* trades bandwidth to reduce run-time; despite this, it only requires 300% the bandwidth and 280% the run-time of *Simple Sync* to guarantee accuracy.

3.3 Comparing *Corrective Syncs*

In the observed simulations, *Simple Monthly Iterative Simple Sync* only outperforms *Simple Monthly Daily Row-Count Sync* when bounded, so in scenarios where backlogging further than one month into the past is unlikely, it is the winner of the "balanced" strategies.

4 Conclusions

Without backlogging, *Simple Sync* is the optimal solution; the variant choice relies on the data stream's frequency, number of IDs, and cost of bandwidth. *Simple Backtrack Sync* is the recommended strategy when a known backlogging interval exists.

[2]This scale was chosen to clearly distinguish the "best" and "worst" strategies.

[3]In scenarios with a 10% backlogging rate

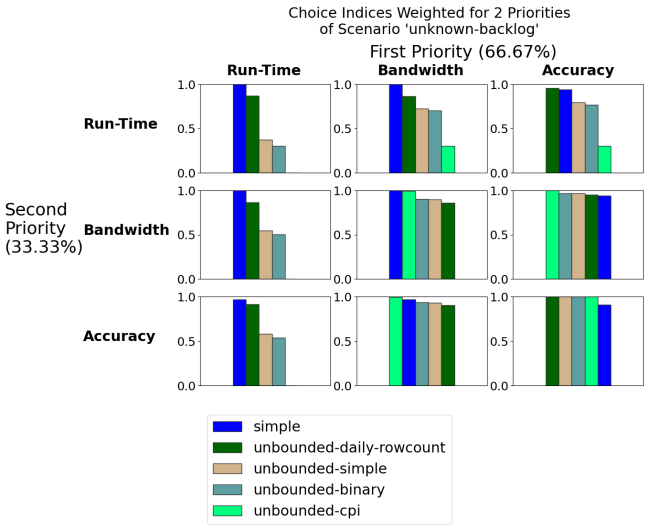

Figure 1: To maintain perfect accuracy, *Iterative Syncs* vary significantly in bandwidth and run-time (normalized, "higher is better").

For scenarios like *unknown-backlog*, the decision depends on the user's priorities. *Daily Row-Count Sync* guarantees perfect accuracy with respectable run-time and bandwidth, and if a small amount of error is tolerated, the bounded variant of the corrective strategy *Simple Monthly Iterative Simple Sync* balances all three metrics. *Iterative CPISync* is only justified in extreme situations where bandwidth is at a premium.

		First Priority		
		Run-Time	**Bandwidth**	**Accuracy**
Second Priority	**Run-Time**	*Simple*	*Simple*	*Daily Row-Count*
	Bandwidth	*Simple*	*Simple Join*	*Iterative CPISync*
	Accuracy	*Simple Monthly Iterative Simple (bounded)*	*Iterative CPISync*	*Daily Row-Count*

Table 1: Strategy recommendation chart for situations similar to *unknown-backlog*

References

Madhu Ahluwalia, Ruchika Gupta, Aryya Gangopadhyay, Yelena Yesha, and Michael McAllister. 2010. Target-Based Database Synchronization. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, page 1643–1647, New York, NY, USA. Association for Computing Machinery.

Chen Tang, Anton Donner, Javier Mulero Chaves, and Muhammad Muhammad. 2010. Performance of Database Synchronization Algorithms via Satellite. In *2010 5th Advanced Satellite Multimedia Systems Conference and the 11th Signal Processing for Space Communications Workshop*, pages 455–461.

Ari Trachtenberg, David Starobinski, and Sachin Agarwal. 2002. Fast PDA Synchronization Using Characteristic Polynomial Interpolation. In *IEEE INFOCOM*, pages 1–10.