Stateful Streaming with External Memory On Workstations

Stony Brook University

Daniel DeLayo ddelayo@cs.stonybrook.edu

Advised by Professor Michael A. Bender, Department of Computer Science

Event Detection in High-Bandwidth Streams

Real-time monitoring of high-rate data streams, with the goal of detecting and preventing malicious events, is a critical component of defense systems for cybersecurity.

Each component **observation** in a suspicious **event** may only occur very rarely and may be separated by billions of unrelated observations. These **low-and-slow** and are notoriously hard to detect.

To detect low-and-slow events, all observations must be stored and processed. Simple logging isn't sufficient, as reports must be timely.

We ingest 4 to 10 million observations per second into RAM and Optane PMEM.

Timely Event Reporting

Timely Event Reporting: Given an endless stream of key-value pairs representing the changes to the state of the system, detect and report keys whenever said key becomes reportable with bounded delay.

The delay is bounded as a function of the **span** of the key; the span of a reportable key is the number of observations between the oldest and youngest observation in the report.

Example Problem: Ingest 5 billions keys-value pairs. One of these keys **k** appears at least twice, separated by one billion key-value pairs.

Goals: Report ***k*** (and *only* ***k***) within 1 billion observations. Ingest millions of keys per second on a workstation. Primarily store data on external memory.

RAM isn't enough for Low-And-Slow events

Detecting a pattern of low-and-slow events is memory-intensive. If the data structures is RAM only, by the time the final observation in an event occurs, it is almost certain that some of the earlier observations have been discarded for lack of space.

Prior work[1] shows that current state-of-the-art solutions report almost no anomalies when the keyspace exceeds RAM for the Firehose benchmark. 2^20 ~ 1 million.

KeyspaceSize	ReportableEvents	ReportedEvents	PercentReported
2^{20}	94368	62317	66%
2^{21}	63673	15168	24%
2^{22}	17063	9	0.0005%

We need to scale effectively into external memory to expand our manageable keyspace.

[1]CLSAC'16, Stateful Streaming in Distributed Memory Supercomputers. Jon Berry and Alexandra Porter

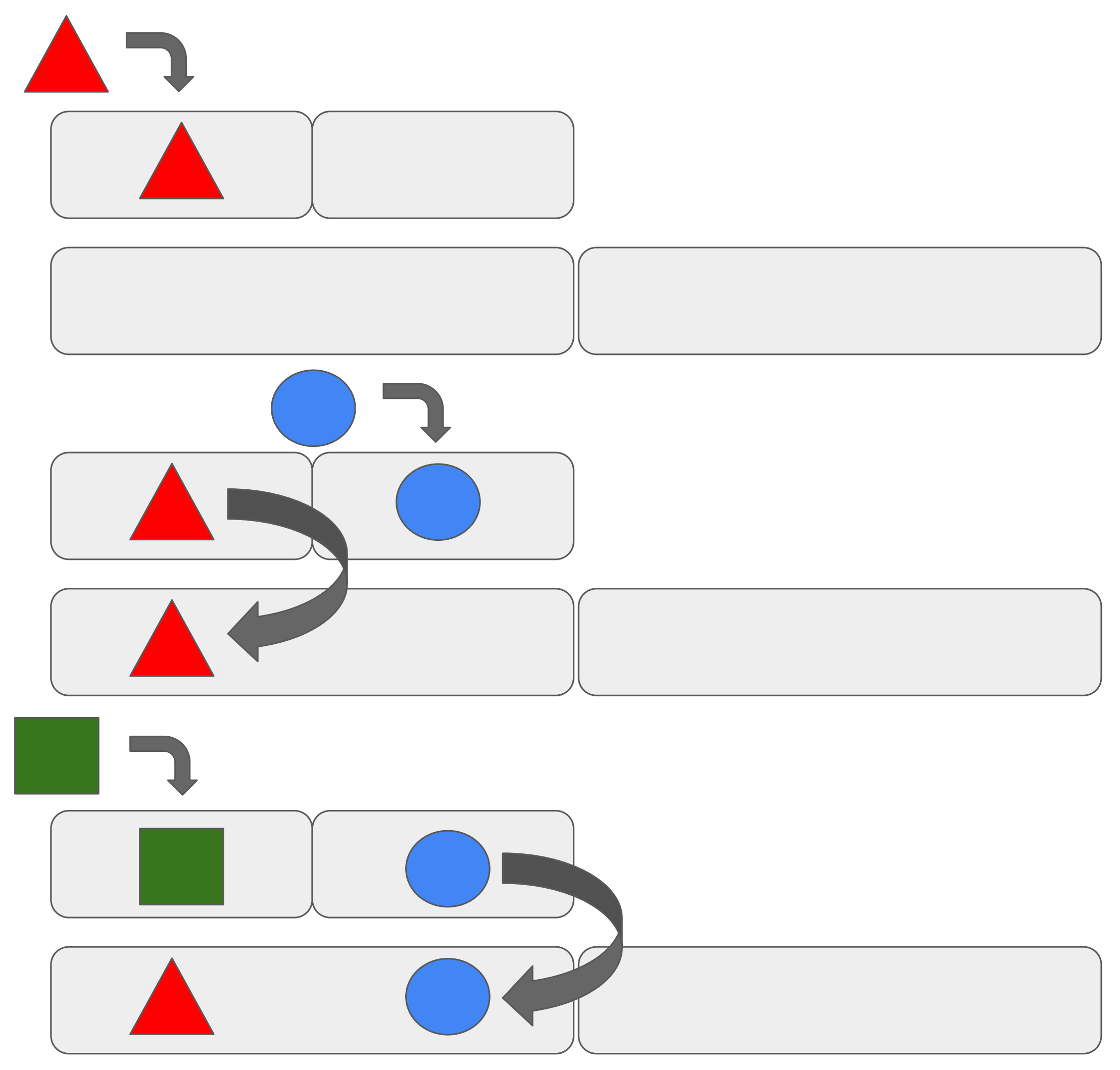
We Scale into External Memory: the LERT

Our solution is the **Leveled External Memory Reporting Tree**, or LERT. A LERT has ***b*** bins per level and each level has bins ***r*** times larger than the previous level.

An object is tracked by its **age** and **level**, or bin and level.

A flush increments an object's age by 1. If it's age reaches *b*, it is moved to the next level instead.

If a flush fills a bin (every *r* flushes into a level), the bin's entire level is flushed.



A flush to level i includes a scan and flush of all previous levels. That is, when flushing to level i, objects are traversed in order regardless of level.

Objects are merged, reported, and possibly flushed to the next level.

If a key is flushed from the bottom level, we simply expire it instead. We call this **Last Bin Expiration**.

That is, only keys in the last bin are subject to expiration.

By accessing external memory in a sequential scan, we're able to process keys stored on external memory at high bandwidth.

With an extra buffer on top, we're able to perform this sequential scan without interrupting insertings to the top level.

Max Gap: LERTs detect Low-And-Slow events

Last Bin Expiration means the oldest bin is up for expiration. We can calculate the minimum number of observations that must be inserted after a key in order for the key to expire. We call this the **Max Gap.**

While flushing, we can (repeatedly) merge keys with younger ones (if available), pushing back expiration until the younger key is expired.

That is, the **Max Gap** determines the longest **interarrival count** between keys before *any* prior state is able to be lost.

The LERTs **Max Gap** is a large fraction of the number of observations it can hold.