Streaming in Practice

Karthik Ramasamy, Maosong Fu, Bill Graham, Vikas Kedigehalli,
Christopher Kellogg, Neng Lu, Sailesh Mittal, Jingwei Wu

Twitter, Inc.

1 Introduction

Twitter is a pioneer and global leader in social media, generating tens of billions of events per hour with over 315 million monthly active users. Analyzing these events to surface relevant content and to derive insights in real time is a challenge. To address this need, we developed Heron, a new real time distributed streaming engine.

Stream processing platforms enable enterprises to extract business value from data in motion similar to batch processing platforms that facilitated the same with data at rest [10]. The goal of stream processing is to enable real time or near real time decision making by providing capabilities to inspect, correlate and analyze data as it flows through data processing pipelines. There is an emerging trend to transition from predominant batch analytics to streaming analytics driven by a combination of the increased data collection in real time and the need to make decisions instantly. Several scenarios in different industries require stream processing capabilities that can process millions and even hundreds of millions of events per second. Twitter is no exception.

Twitter is synonymous with real time. When a user tweets, his or her tweet can reach millions of users instantly. Twitter users post several hundred millions of tweets every day. These tweets vary in diversity of content [3] including but not limited to news, pass along (information or URL sharing), status updates (daily chatter), and real time conversations surrounding events such as the Super Bowl, the Oscars, etc. Due to the volume and variety of tweets, it is necessary to surface the relevant content in the form of break out moments and trending #hashtags to users in real time. In addition, there are several use cases in real time such as analyzing user engagements, extract/transform/load (ETL), model building, etc.

In order to power the aforementioned crucial use cases, Twitter developed an entirely new real time distributed stream processing engine called Heron. Heron is designed to provide

- **Ease of Development and Troubleshooting**: Users can easily debug and identify the issues in their topologies (aka standing queries), allowing them to iterate fast during development. This is possible because of the fundamental change in architecture in Heron from thread based to process based. User can easily reason about how their topologies work and profile/debug its components in isolation.

- **Efficiency and Performance**: Heron is 2x-5x more efficient than Storm [8]. This resulted in significant cost savings for Twitter both in capital and operational expenditure.

- **Scalability and Reliability**: Heron is highly scalable both in the ability to execute large number of components for each topology and the ability to launch and track large numbers of topologies. Such a large scale is achieved because of the clean separation of topology scheduling and monitoring.

- **Compatibility with Storm**: Heron is API compatible with Storm and hence no code change is required for migration.

- **Simplified and Responsive UI**: Heron UI gives a visual overview of each topology. The UI uses metrics to show at a glance where the hot spots are and provides detailed counters for tracking the progress and troubleshooting.

- **Capacity Allocation and Management**: Users can take a topology from development to production in a shared cluster infrastructure instantly since Heron runs as yet another framework of the scheduler that manages capacity allocation.

2 Heron Data Model

Heron uses a directed acyclic graph (DAG) for representing a real time computation. The graph is referred to as a topology. Each node in the topology contains the processing logic and the links between the nodes indicate how the data flows between them. These data flows are called streams. A stream is an unbounded sequence of tuples. Nodes take one or more streams and transform them into one or more new new streams. There are two types of nodes: spouts and bolts. Spouts are the sources of streams. For example, a Kafka [4] spout can tap into a Kafka queue and emit it as a stream. A bolt consumes tuples from streams, applies the processing logic and emits tuples in outgoing streams. Typical processing logic includes filtering, joining and aggregation of streams.

In this topology, the spouts S1 taps into its data source and injects two streams consumed by the first stage bolts B1, and B2. These bolts transform the streams and emit three new streams feeding bolts B3 and B4. Since the incoming data rate might be higher than the processing capability of a single process or even a single machine, each spout and bolt of the topology is run on multiple tasks. The number of tasks for each spout and bolt is specified in the topology configuration by the programmer. Such a task specification is referred to as the degree of parallelism.

3 Heron in Practice

3.1 Back Pressure

Spout based back pressure helped us reduce the data loss significanlty as stragglers are the norm in a multi-tenant distributed systems. Heron back pressure recovery mechanism allows us to process data at a maximum rate such the recovery times are very low. Since most topologies are provisioned with extra capacity to handle increased traffic during well-known events (like the Super Bowl, the Oscars etc.), the recovery rate is usually much higher than the steady state. In cases where the topologies have not been provisioned to handle increased traffic, the back pressure mechanism acts as a shock absorber to handle any temporary spikes. In cases where these spikes are not temporary, back pressure also allows users to add more capacity and restart their topologies with minimal loss of data.

We have encouraged topology writers to test their back pressure (and recovery) mechanism in staging environments by artificially creating traffic spikes (e.g., by reading from older offsets in Kafka). This allows them to understand the dynamic behavior of the back pressure and measure the recovery time. To monitor this process in real time, several metrics have been exposed on the dashboard. Back pressure also helps topology writers in tuning their topology.

In our experience, we have found that in most of the scenarios back pressure recovers without manual intervention. While most users see back pressure as a requirement, some users prefer dropping of data as they only care about the latest data. To handle such cases, we added the load shedding feature in spouts as decribed in the following section.

3.2 Load Shedding

Load shedding has been studied extensively in the context of second generation streaming systems [6, 1, 7, 9, 2, 5]. Most of the proposed alternatives broadly fall into two broad categories, sampling based approaches and dropping based approaches. The idea behind sampling based approaches is that if the system can automatically downsample an incoming stream in a predictable way, the user can potentially scale up the results of the computation in order to compensate for this.

Due to added complexity in sampling, Heron currently implements a dropping-based approach that will drop older data and prefer more recent data when the Heron topology is unable to keep up. Heron spouts allow configuring a *lag threshold* and a *lag adjustment value*. The lag threshold will indicate how much lag is tolerable before the spout drops any data. The lag adjustment value will indicate how much of the old data the system will drop when this threshold is reached.

Given the lag threshold and lag adjustment value configuration, the system will monitor the lag for each individual spout instance and periodically skip ahead by the lag adjustment value whenever the lag is above the threshold value. A key point here is that the decision to drop data is a completely local decision in each spout instance. There will be no attempt made to synchronize amongst different spouts or otherwise coordinate such that the spouts work together in deciding what data to drop.

4 Acknowledgements

Thanks to David Maier and Kristin Tufte for providing comments on the initial draft of the paper that helped improved its presentation. Thanks to Jeff Naughton, Deep Medhi and Jignesh Patel for reading the pre-final draft and help improve the presentation. Thanks to Arun Kejariwal for help with LaTeX including setting it up, patiently answering several questions and providing several comments on the first draft.

References

[1] B. Babcock, M. Datar, and R. Motwani. Load shedding techniques for data stream systems. In *In Proc. of the 2003 Workshop on Management and Processing of Data Streams (MPDS*, 2003.

[2] B. Babcock, M. Datar, and R. Motwani. Load shedding in data stream systems. In C. Aggarwal, editor, *Data Streams*, volume 31 of *Advances in Database Systems*, pages 127–147. Springer US, 2007.

[3] S. Dann. Twitter content classification. *First Monday*, 15(12), December 2010. http://firstmonday.org/ojs/index.php/fm/article/view/2745/2681.

[4] N. N. Jay Kreps and J. Rao. Kafka: A distributed messaging system for log processing. In *SIGMOD Workshop on Networking Meets Databases*, 2011.

[5] S. Senthamilarasu and M. Hemalatha. Article: Load shedding using window aggregation queries on data streams. *International Journal of Computer Applications*, 54(9):42–49, September 2012.

[6] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 309–320, 2003.

[7] N. Tatbul and S. Zdonik. Window-aware Load Shedding for Aggregation Queries over Data Streams. In *International Conference on Very Large Data Bases (VLDB'06)*, Seoul, Korea, September 2006.

[8] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy. Storm@twitter. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 147–156, 2014.

[9] Y.-C. Tu, S. Liu, S. Prabhakar, and B. Yao. Load shedding in stream databases: A control-based approach. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 787–798, 2006.

[10] J. Vijayan. Streaming Analytics: Business Value from Real-Time Data. http://www.datamation.com/data-center/streaming-analytics-business-value-from-real-time-data.html.