Introduction

The escalating growth of unstructured data is leaving organizations unable to make critical decisions in time to realize business benefit. Complex machine-generated data, log files, and data from mobile and gaming apps are all growing at unprecedented rates, and arriving faster than ever. Today, many firms are simply unable to ingest or process the massive amount of data that is being generated around their business in a timely manner, leaving them unable to take meaningful action. This reality is at odds with the need to provide time-critical responses to risks and threats as well as exploiting opportunities that can make a positive difference to the bottom line.

Organizations need to make decisions in “business time”—the speed at which their business runs today—with most expecting to make faster and more insightful decisions faster in the future. Unfortunately, most organizations today are making low-impact or no-impact decisions based on small sets of data. More importantly, the lag in traditional (often batch-oriented) big data processing means that decisions often come hours or days after the value of that data to the business has decreased or vanished. Forester rightly calls these important opportunities “perishable insights”, and exploiting them requires a streaming analytics platform capable of detecting and acting on potentially complex events in high-velocity streams of data.

Enterprises today are looking to Hadoop as the core of their big data strategy. However, the traditional MapReduce paradigm is not meeting business needs for business-time awareness and actionable insights. Despite the ability to do real-time queries in Hadoop, the data must first go through a batch process before queries can be made. For many enterprises, this time lag is simply too long for real value to be extracted. A streaming analytics platform provides an end-to-end solution that takes data directly from multiple sources in real time, combines it with existing data, and delivers actionable analysis. As such, the requirements for a streaming analytics platform to address these issues include the following:
BIG DATA NATIVE. Streaming analytics is fundamentally a big data problem. Effective systems must natively integrate with and extend the capabilities of big data platforms such as Hadoop 2.0, without requiring add-on mechanisms that generate complexity or compromise reliability.

HIGH THROUGHPUT, LOW-LATENCY PROCESSING. Given the nature of the tasks at hand, the platform should be able to handle throughput up to billions of events per second. Systems must be fundamentally nonblocking, without cumbersome acknowledgement mechanisms that are unable to scale.

SCALABILITY IN BOTH DATA INGESTION AND DATA PROCESSING THROUGHPUT. All data streams should be expected to grow, and the platform ought to be able to scale linearly with them. Any substantial consistent increase in data volume, variety, or velocity should be facilitated by adding more nodes to the cluster with minimal diminishing returns.

STATEFUL FAULT TOLERANCE. There are bound to be node outages and service disruptions in any distributed computing environment. The platform should be able to automatically compensate for these occurrences without data loss, state loss, or human intervention at both the system and the application level. A robust checkpointing and recovery mechanism is an absolute requirement.

EASY APPLICATION CREATION, DEPLOYMENT, AND OPERATION. Independent of their roles, level-of-business IT, data scientists, and developers all need the ability to create and run applications easily. Moreover, long-lasting, successful applications can only be possible if they are easy to write and maintain. The programming, deployment, and operational model should allow users to focus on their business problem, without unnecessary complexity.

DataTorrent RTS enables the high-performance and complex processing of data as it is generated—providing the ability to take meaningful actions in real time. Capable of processing billions of events per second and enabling the ability to take action in real time, DataTorrent RTS was designed from the outset for the rigors of streaming analytics, meeting all of the criteria described above and more. With a unique and innovative architecture, the platform provides scalable data ingestion from any source, coupled with high performance low-latency complex data processing that delivers business-time actions and alerting. With DataTorrent RTS, organizations can utilize their existing big data environments and processes to deliver dynamic execution and fault tolerant streaming processing that is easy to program, deploy, and use.

Real-Time Streaming Analytics with DataTorrent RTS

Today information is coming from multiple disparate sources, and it is all arriving faster than ever. Increasing mobility, clickstreams, proliferate sensors, and the growing Internet of Things (IoT) are generating immense and relentless quantities of data. Making sense of this data and deriving valuable insights requires a truly capable streaming analytics platform.

Delivering Actionable, Business-Time Insights

More than simply accumulating data and analyzing it after the fact, organizations need ways to process data in as it is generated, acting within a window of opportunity before insights quickly lose their value. Streaming analytics platforms are designed to facilitate the extraction of perishable insights, allowing organizations to:

- Detect events that drive risks
- Exploit opportunities in real time
- Understand complex environments visually
- Automate immediate responses

In an already dynamic big data marketplace, streaming analytics is growing very rapidly. There has been a 66% increase in streaming analytics in the past two years. This growth is driven by a fundamental need to understand both risks and opportunities as early as possible in order to respond.

“Exploiting Perishable Insights is a Huge, Untapped Opportunity for Firms: Forrester defines perishable insights as urgent business situations (risks and opportunities) that firms can only detect and act on at a moment’s notice. Streaming analytics platforms can help firms detect such insights in high velocity streams of data and act on them in real-time.”

DataTorrent RTS

Streaming analytics is all about time—time to ingest data, time to process data, and time to take action based on the insights gained. DataTorrent RTS is different than other streaming analytics solutions because it is fundamentally architected around the concept of time. Systems that are based on events (e.g. Storm) must acknowledge each event, resulting in architectural inefficiencies that can hamper scalability. Systems that are based on MapReduce (e.g., Spark) introduce overhead of batching the data, adding latency. By using the concept of time for its advanced windowing processing capabilities. DataTorrent RTS requires no complex event acknowledgement infrastructure, instead using streaming window event processing as its core mechanism.

- In DataTorrent RTS, a time-based core system window is typically one half second, and the system fundamentally doesn’t care if 10 events or a million take place inside that half-second time window.
- Bookkeeping (including fault tolerant check-pointing) is conducted as a multiple of the core window time (typically every thirty seconds).
- Application windows are provided on top of the core system window time, allowing programmers to have sliding windows or time-series windows, as befits their application.

The innovative DataTorrent RTS architecture lets organizations harness the full potential of real-time big data by taking immediate action. As described in detail in this paper, DataTorrent RTS delivers the following capabilities:

- OPEN HADOOP NATIVE PLATFORM. Deployable either on premise or in the cloud, DataTorrent RTS is a Hadoop 2.0-native application that is certified on all major Hadoop distributions. RESTful API is provided for management flexibility, letting organizations chose the management tools that make sense for them. Over 400 fully functional open source Java-based operators are already provided under Apache 2.0 license in the Malhar repository.

- STREAMING WINDOW EVENT PROCESSING. DataTorrent RTS provides built-in streaming window event processing, supporting streaming event windows, and both aggregate and sliding application windows. Through this mechanism, the system offers comprehensive processing guarantees and no throughput-versus-latency trade-off.

- IN-MEMORY DATA PROCESSING. In-memory data processing provides high-performance complex event processing with support for billions of events per second. Highly optimized event-window processing delivers latency in the milliseconds, providing considerable headroom for new demands. All events are processed strictly in order in lossless fashion—even across system failures—guaranteeing that data and state are never lost.

- FAULT TOLERANT, DYNAMIC EXECUTION. Unlike other streaming analytics solutions, DataTorrent RTS provides operational stability with a system-level fault-tolerant architecture that ensures recovery from hardware and software errors. Stateful fault tolerance at the application level helps ensure that computations persist across system failure. Moreover, streaming applications can be modified on a running system to better support business decisions—without downtime.

- SCALABLE, ANY-DATA ANY-SOURCE INGESTION. DataTorrent RTS is able to efficiently ingest both structured and unstructured data seamlessly, and is able to accommodate data fluctuations in size, form, and frequency. Over 75 existing input adaptors are a part of the open source Malhar operator library, and full-featured templates allow new adapters to be created in hours. The system autoscales and partitions dynamically to meet ingestion SLAs for the throughput and the latency while underlying fault tolerance allows for input mechanism or component failure with fully stateful restarts.

- REAL-TIME ACTIONS AND ALERTING. Built-in system-level and application-level windows enable monitoring and alerting for both time-series and rolling time data. Flexible Java-based operators provide a familiar programming model and enable complex business logic to automate actions based on events. Events and notifications can be delivered to the external console or applications via Java Message Service (JMS), WebSocket, and other mechanisms. Events and notifications are also integrated with external systems such as SMS and SMTP gateways for alerting on critical events.

- EASY TO USE AND OPERATE. With a familiar programming model and a choice of Java-based integrated development environments (IDEs), developers can easily create, debug, and package streaming applications. Level-of-business IT and technical data scientists can construct and configure applications with DataTorrent’s graphical application and assembly tool. Data center operations are simplified with a full-featured system management console that provides system health and statistics via GUI and CLI. A visual real-time dashboard designer simplifies custom dashboard creation.
Anatomy of a Streaming Application

In order to fully appreciate the benefits of the DataTorrent RTS architecture, it is important to understand how a streaming application is defined. Despite its benefits, streaming analytics does not pose a radically different approach to application development. The reality is that thinking in streams and programming for DataTorrent RTS is remarkably straightforward and familiar using industry-standard Java API calls.

Working with Data Flow Diagrams

Streaming analytics programming models are centered around the data stream itself, a paradigm that reverses traditional programming models. With traditional models, code execution controls data—data is at rest, and queries against that data are active. In DataTorrent RTS, the relationship is reversed. Incoming data controls the execution of the code. In other words, the query is at rest, and data flows through it. As a result, applications take the form of a data flow diagram, also known as a directed acyclic graph (DAG, Figure 1).

In this programming model, the data flow diagram is comprised of streams and operators. A stream is a sequence of data tuples with schema. Operators take one or more streams as input, perform computations, and emit one or more output streams. Each operator is comprised of business logic expressed in Java, either drawn from the open source Malhar library, or custom written. Operators have at least one port (input or output), and there is no upper limit on the number of ports that an operator can have. Operators with no input port are called input adapters, as they consume data from outside the application. All operators can access outside sources of data, so applications can read and write data from outside of the Hadoop cluster via any operator.

Architectural Overview

A high-level architectural perspective of DataTorrent RTS is provided in Figure 2. The entire system functions as a Hadoop 2.0-native application, running on top of the Hadoop YARN (yet another resource negotiator) mechanism. All Hadoop 2.0 distributions are supported, and DataTorrent RTS is fully enabled for multitenancy so that it can coexist with MapReduce and other Hadoop jobs running on the cluster. To facilitate this integration, DataTorrent provides a Streaming Application Master—StrAM—a native YARN application master that serves as the brain and the bookkeeper of the streaming application. StrAM is responsible for managing the DataTorrent RTS application within the Hadoop cluster, and for managing DataTorrent real-time processing functionality.

Figure 2. DataTorrent RTS is implemented as a Hadoop 2.0 native application that works with the open source Malhar operator library.

Directed Acyclic Graph (DAG)

Figure 1. DataTorrent RTS applications take the form of a data flow diagram, with operators that essentially filter incoming streams.
Extensible and Open DataTorrent RTS Operators

Operators are the fundamental building blocks used by streaming analytics systems. DataTorrent RTS applications can be built from over 400 operators in the open source Malhar library (https://github.com/DataTorrent/Malhar). Custom operators can also be easily written, using the Malhar operators as templates. Figure 3 illustrates the broad categories of operators available in the Malhar library. In addition to operators, the Malhar repository contains:

- Benchmarks for performance testing applications
- Additional operators contributed for interfacing with third party software
- Sample code that demonstrates how to use the library operators.

Figure 3. The open source Malhar library contains over 400 operators facilitating most applications.

More than mere examples, all Malhar operators are fully functional and capable of taking advantage of advanced DataTorrent RTS functionality. Each operator includes the following capabilities in addition to the core business logic:

- **FAULT TOLERANCE.** Malhar operators can use the platform's checkpoint capability to ensure that there is no data loss under any failure scenario.
- **PROCESSING GUARANTEES.** All processing guarantees are supported, as applicable to the operator.
- **DYNAMIC UPDATES.** Any operator property can be changed on the fly.
- **EASE OF EXTENSIBILITY.** All Malhar operators are based on templates that are easy to extend and customize.

Native Hadoop Integration

As the de facto big data platform, Hadoop has solved many of the key big data challenges and it has considerable traction in terms of deployed clusters as well as a compelling ecosystem of analytics tools. Rather than creating yet another custom siloed infrastructure solution, DataTorrent RTS is implemented directly on Hadoop 2.0 as a fully native application, using the native Hadoop YARN scheduler.

Many competing streaming analytics solutions use a bolt-on or hybrid approach. While some employ Hadoop as a portion of their solution, critical elements like schedulers are often implemented outside of Hadoop, limiting both function and performance while increasing costs for maintenance and operation. This fragmented approach can cause a number of issues:

- Conflicts can arise between the Hadoop scheduler (YARN) and any external scheduler.
- Multitenancy issues can result, making it difficult to have multiple jobs running on the cluster simultaneously.
- A hybrid approach also creates unnecessary operational complexity, and misses the opportunity to utilize native Hadoop capabilities such as HDFS to their full potential.

In contrast, DataTorrent RTS applications run natively on Hadoop 2.0. In fact, YARN does not differentiate between a MapReduce job and a DataTorrent RTS streaming analytics application. In this sense, a DataTorrent RTS application is just another application running on the Hadoop 2.0 cluster. In addition, DataTorrent RTS is completely agnostic to the Hadoop 2.0 distribution, and has been certified on all major distributions, including those from Amazon EMR, Cloudera, Hortonworks, MapR, and Pivotal.

DataTorrent RTS runs in either new or existing Hadoop 2.0 deployments. All investments that utilize multi-tenancy, security, data flow integration, monitoring, metrics collection, and so on will require no changes when streaming applications are run on Hadoop. This approach presents a number of distinct advantages:

- DataTorrent jobs can exploit Hadoop functionality such as writing to HDFS as needed to checkpoint and record data, or running the application on a specific job queue, or honoring the Kerberos security model.
Multi-tenancy is automatic, in that multiple DataTorrent jobs and regular Hadoop jobs simultaneously share the cluster.

Being Hadoop native also facilitates subsequent data post-processing and analysis from the Hadoop ecosystem of tools.

DataTorrent RTS was designed from the outset to run on YARN to take advantage of both its resource management and scheduling capabilities. YARN’s scheduler is generic, allowing other kinds of applications to utilize the resources of the Hadoop cluster alongside MapReduce. DataTorrent RTS uses YARN to extend Hadoop into a real-time streaming platform.

The ability to access HDFS from DataTorrent RTS is particularly important. HDFS helps provide high availability for StrAM—the YARN application master for the DataTorrent RTS streaming application. Checkpoints recording application state are also written to HDFS, allowing DataTorrent RTS to recover from errors and outages without data loss. HDFS also eliminates the need for external storage, which would otherwise cause scale issues and operational overhead.

To understand how DataTorrent RTS works, it is helpful to think in terms of multiple plans that define the application at different stages.

- **The logical plan** consists of defining the data flow and business logic.
- **The physical plan** determines the resources needed to execute the logical plan.
- **The execution plan** maps resource requirements to available system components.

The sections that follow detail the process by which DataTorrent RTS converts a data flow diagram into a streaming application executing on a Hadoop 2.0 cluster.

### Logical Plan

The first step in developing a DataTorrent RTS application is creating a logical plan—or development workflow—by adding operators and streams to a DAG. To do this, developers work in their choice of Java IDE, including NetBeans, Eclipse, IntelliJ, etc. Developers can use operators from the Malhar operator library as well as write their own custom operators.

The logical plan consists of one or more input sources connected via the DAG to one or more output destinations (Figure 4). This graph of operators connected by streams—together with their functional state—comprises the application topology. Importantly, the attributes that impact the physical layout of the application are attached to the logical plan, but only applied later during the creation of a physical plan.

### Physical Plan

The physical plan represents a real-time runtime execution workflow that is ready to run on a Hadoop 2.0 cluster. The physical plan is defined as the result of the transformation of the logical plan by applying all of the attributes that specify the physical nature of the DAG. In this process, the logical plan is compiled by the platform into a physical plan (Figure 5), creating one or more physical operators per logical operator. Physical operators are ready to run on a Hadoop 2.0 cluster.

As shown in Figure 5, physical operators can be configured to run on any physical node in the Hadoop cluster.
Physical operator instances can also be automatically created or destroyed at runtime if ingest rates or load increases or decreases. StrAM starts the application by configuring atomic processing windows with a default of .5 seconds (500 ms) duration with the input adapters.

StrAM communicates with the YARN resource manager, and serves as the bookkeeper of the streaming application. Multi-tenancy is accomplished by having multiple StrAMs communicating with the YARN resource manager, one for each DataTorrent RTS application sharing the cluster.

As pictured in the illustration, The DataTorrent Gateway is a service that communicates with the YARN Resource Manager, exposing a RESTful API and WebSocket communication for cluster and application information. The DataTorrent Console communicates with the gateway to manage all DataTorrent applications and also supports tasks needed to deploy, debug, and monitor applications.

**Execution Plan**

The execution plan is defined as a result of mapping the physical plan to the resources made available by the YARN resource manager. As a native YARN application master, StrAM is responsible for the application launch, resource allocation, partitioning, and scheduling of the streaming application to be executed on the Hadoop cluster. StrAM is also responsible for web services, run-time changes, statistics, SLA enforcement, security, purging old unneeded state, and node outage recovery. Depending on how many Hadoop containers are available, StrAM maps the physical plan into an execution plan, using the YARN Resource Manager and Node Managers to deploy the DAG onto the cluster. At this point all of the operators are mapped to specific Hadoop containers, and the application is ready to run. Hadoop remote procedure call (RPC) is used communicate between the components.
the YARN Application Master and a container manager process running within each Hadoop container.

Figure 6 illustrates a simplified high-level perspective of a DataTorrent RTS application running on a Hadoop 2.0 cluster, and sharing cluster resources with traditional Hadoop MapReduce jobs. StrAM is a native Hadoop 2.0 application that is deployed onto the cluster just like any other application while physical operators are deployed into containers scheduled across the cluster by the YARN resource Manager. The YARN resource manager communicates with the Hadoop Node Manager running on each physical node in the Hadoop cluster. StrAM in turn communicates with each Hadoop container that hosts a physical operator. Note that one or more operators can operate within a single Hadoop container at the discretion of the developer (not shown).

The stream proceeds from operator to operator as determined by the application. A Bufferserver in each container manages the physical streams of the container for ports that have readers outside of the container. If the message queue grows too long, the Bufferserver simply spools to a local disk. A command-line interface (CLI) communicates with the YARN resource manager, allowing the user to access and change applications. The CLI encapsulates access to the YARN and webservice APIs.

Streaming Window Event Processing

Streaming applications have special needs in terms of both operability and functionality. In terms of operability, applications need bookkeeping, fault tolerance, and state management. From a functional standpoint, applications need to be able to accumulate data in a manner that fits with specific business needs. In DataTorrent RTS, both operability and functionality constraints are accommodated with streaming window event processing.

Built-in Windowing Support

Windowing is a fundamental native capability of the DataTorrent RTS streaming platform. As implemented in DataTorrent RTS, streaming windows are finite time slices that constitute a large set of consecutive tuples (Figure 7). The system is nonblocking and no acknowledgements are necessary. Windowing results in sophisticated abilities to enable and boost linear scalability as well as to provide for inherent fault tolerance as a fundamental system capability.

Streaming windows and two types of application windows are provided, giving developers considerable flexibility.

- **STREAMING EVENT WINDOWS.** In DataTorrent RTS, streaming windows are a natively supported paradigm. A streaming window is a finite slice of time that encapsulates a set of tuples. The implementation is nonblocking and all bookkeeping operations are done by the platform between the end of one window and the start of the next. The default window is 500 ms, and that time constitutes the minimal amount of computation that would need to be redone in an error situation. There is only one streaming window period for an application.
**AGGREGATE APPLICATION WINDOWS.** Charting ticker data on a per-minute basis is an example of an aggregate application window. An aggregate application window is constructed by combining multiple consecutive streaming windows in a non-overlapping fashion. The next aggregate application window starts only after the current one finishes. This means that there is no overlap between two application windows of this operator. Upon the completion of the aggregate window, all of the state of the operator can be flushed.

**SLIDING APPLICATION WINDOWS.** Counting the top 10 trades in terms of volume on a ticker over the past minute is an example of a stateful sliding application window. A sliding application window is constructed by combining multiple consecutive streaming windows in an overlapping fashion. The next sliding application window drops the last streaming window of the previous sliding window, and picks up the new upcoming streaming window to form a new application window. In effect, each sliding window slides by one streaming window.

The system provides out-of-the-box support for multiple application windows in the same application. Guaranteed event processing is assumed, with no positive acknowledgement provided or needed. Event sequence is guaranteed via TCP, and the windowing architecture itself enforces the notion of time. DataTorrent RTS operators are scheduled one at a time and designed to never stop.

### Comprehensive Processing Guarantees

Processing guarantees are essential to assure developers that their applications are functioning as intended, and that data isn't either lost, or possibly processed multiple times in the event of failures and replays. Other solutions provide only “at least once” processing guarantees, and have to be extended to provide “exactly once” guarantees, often adding to the latency in the process.

Using streaming window event processing, DataTorrent RTS guarantees that every tuple will be fully processed. The system further provides the semantics of all three processing guarantees as a part of the basic platform to accommodate multiple application models:

- **EXACTLY ONCE.** Each tuple will be processed exactly once.
- **AT LEAST ONCE.** Each tuple will be processed at least once. Note that this is stateful processing so the processed results are identical to the exactly one semantics.
- **AT MOST ONCE.** Each tuple will be processed at most once. Here we are intentionally dropping the events to maintain the freshness of the results.

### No Throughput vs. Latency Tradeoff

Other stream processing solutions force a choice between tuple-at-a-time processing for low latency, or “micro-batch” processing for higher throughput.
In contrast, DataTorrent RTS offers the ability to achieve extremely high throughput while guaranteeing low latency, all with tuple-at-a-time processing.

Additionally, developers don’t have to compromise between available processing guarantees since the business logic implemented by the operators works orthogonal to these guarantees which are operational attributes over the operators.

**In-Memory Processing for Performance and Low Latency**

By their nature, big data streaming analytics applications require high performance and low latency. Traditional big data analytics platforms work by reading previously written data from spinning storage media, a process that does not lend itself to either performance or low latency. In contrast, DataTorrent RTS employs real-time computation. Processing is performed on streams in memory, with asynchronous checkpointing to HDFS for state preservation. Through this architecture, Hadoop is transformed into a real-time distributed operating system, and applications can consume and produce data in real time.

As described, each Hadoop node can host one or more YARN containers. The Container Manager within each container communicates with StrAM, and spawns threads as necessary within the container to process one or more operators. The Bufferserver within each container is then responsible for managing the message queues, allowing for nonblocking throughput. The Bufferserver is implemented as a distributed message queue by the platform with one Bufferserver per container.

Operators and containers within the system work on a publication/subscription (or pub-sub) model. Streams are initialized by the writer port, and read by any number of reader ports, coordinated by the Bufferserver in each container. The writing port writes the tuples in nonblocking fashion. This attribute is critical, as it allows the operator to compute all of the incoming tuples without being blocked by downstream readers. Moreover, if the stream has multiple readers, each can continue to read without being blocked by other readers. The Bufferserver retains the tuples as atomic windows.

**Stateful Fault Tolerance**

Fault tolerance should be basic functionality for any enterprise-ready streaming analytics platform. Both system-level fault tolerance and application-level fault tolerance are essential for seamless operations. To achieve this level of fault tolerance requires key system functionality, including:

- **ENSURING THAT EVENTS ARE PROCESSED AS GUARANTEED.** Events must be processed as stated in the processing guarantees across a system or node outage. Most streaming solutions require that the tuples being processed by the failed node be re-transmitted. In this process they risk losing the order of the stream and also add latency and reduce throughput. DataTorrent RTS uses a recovery window to help with retransmissions of only the failed window. This approach preserves stream order and ensures that there is little to no impact on throughput and latency SLAs.

- **ENSURING THAT APPLICATION STATE IS MAINTAINED.** DataTorrent RTS provides the ability to recover state at the exact point before failure. Checkpointing records the latest application state to HDFS with a default checkpoint window of 60 seconds. The interval is easily configurable based on application needs. Again, all state recovery is done in order. In the case of a failed Hadoop container or node, the state is read from HDFS, indicating the last executed window. The operators from the failed Hadoop node can then be restarted in another container—possibly on another node in the Hadoop cluster—with processing guarantees maintained throughout. Other streaming solutions require complicated hand coding to accomplish fault tolerance, or they are forced to back memory with an external RDMS or other data store to track windows.

- **ENSURING THAT THE PLATFORM IS FAULT TOLERANT.** In order to truly recover from faults with minimal impact to throughput or latency SLAs, a streaming analytics solution needs to be able to re-distribute processes from the failed nodes to other healthy nodes. DataTorrent RTS provides a fault-tolerant StrAM application manager which checkpoints itself, and records its contents to memory. Other solutions provide no support for stateful fault tolerance in the base platform, leaving its implementation as a complex exercise for application developers.

**Dynamic, Scalable, and Fault Tolerant Data Ingestion**

Many streaming applications demand a very high ingestion rate, or may have highly variable ingestion rates. The same
applications may have stringent demands in terms of throughput or latency. The ability to partition and unify a stream is essential in streaming analytics, given the need to scale and adapt to constantly changing conditions and stream rates. Other streaming solutions support only simple field-based partitioning, making them susceptible to input data skews. Implementing stream partitioning in these systems requires hand coding and these platforms also likewise lack a unification mechanism.

DataTorrent RTS implements scalable stream partitioning and unification (Figure 8), with the ability of an input data stream to be split into multiple sub-streams. To accomplish this, multiple instances of operators can be dynamically generated to create multiple parallel processing pipelines in the same DAG. The outputs of these operators are then unified. Splitting criteria are configurable, and can range from criteria such as:

- A latency SLA
- A field in the input record
- Load or throughput

The partitioning and unification process is managed by StrAM, which makes partitioning decisions as a response to the application requests at runtime. StrAM then manages the partitions—creating or destroying containers as needed to meet run-time requirements. Partitioning can also be changed at runtime via the command line or web interface. DataTorrent RTS partitioning and unification provide sophisticated capabilities, including:

- **AUTOSCALING.** The system can automatically scale up or down based on various criteria such as throughput and latency SLAs, with linear scalability due to minimal required bookkeeping.
- **DIMENSIONAL COMPUTATION.** Dimensional computation involves pre-computing every possible query on an event, allowing an analyst to quickly change their query and not have to recalculate the answer. In traditional computing models, this is very complex, requiring an exponential number of calculations. DataTorrent provides this functionality out of the box, providing flexible options that include tuple hashcode, tuple field, or custom rules.
- **INGEST SCALING.** The system can scale dynamically add more input adapters as needed to meet ingestion SLAs for throughput and latency, giving back adapters when peak load has passed. For example, the Kafka or Flume operators can automatically scale up or down based on changes in the number of Kafka partitions or Flume channels.

**Simplified Development, Deployment, and Operation**

For any streaming analytics system to be useful, applications must be easy to develop and deploy for the broadest possible segment of the user population.

DataTorrent RTS has taken a comprehensive approach to these important aspects of the system.
Developers write single-threaded code in Java using their favorite IDE while the system provides all of the necessary parallelization.

Data scientists use an application builder to create apps without coding.

Line of business analysts or users create real-time dashboards with an easy-to-use dashboard builder.

DataTorrent 2.0 now offers the ability to combine Java code objects that allow any data source and business logic for real-time computations using rapid application development, deployment, and monitoring tools.

**Graphical Application Assembly and Deployment**

Real-time streaming applications today are the exclusive domain of knowledgeable technical developers. With DataTorrent RTS graphical application assembly and deployment, users no longer need Java expertise to build real-time streaming applications. The tool combines the ease of visual development tools with the comprehensive data analysis provided by DataTorrent RTS into a simple, easy-to-use web-based application builder. This approach offers easy streaming application assembly for both the developer and the non-developer alike.

Drawing on operators in both enterprise and open source libraries, users can dynamically construct streaming applications, making connections between operators with the click of a mouse (Figure 9). Type checking is automatically performed for operator compatibility. Users have the ability to dynamically set properties, and the application includes the integrated capability to deploy and launch applications.

**Visual Real-Time Dashboard Designer**

For big data applications, true value is defined by the ability to gather key insights in real-time via a visual dashboard. Unfortunately, many dashboards are fixed in purpose and lack the necessary flexibility to be generally useful. With the DataTorrent RTS visual real-time dashboard designer, data scientists now have a palate of visualization tools, allowing them to design a dashboard—in real time, on a running application—via a web-based dashboard builder.

DataTorrent RTS includes a true visual real-time dashboard designer that lets users rapidly build dynamic dashboards (Figure 10). The application generates widgets automatically on running streaming applications, saving considerable time. Users can dynamically change data sources and multiple dashboards can be derived from the same streaming application. This capability is ideal

---

**Figure 9.** DataTorrent’s RTS provides easy-to-use graphical application assembly and deployment.
for providing dashboards that need to serve users with different interests from the same data stream. Support is provided for both Kafka and WebSockets.

Conclusion

As the amount of user-generated and machine-generated data increases, streaming analytics is getting more essential to help organizations gain actionable intelligence while the data still has value to the business. DataTorrent RTS is designed to let organizations take action in real time as a result of high-performance complex data processing, as the data is created. By deploying natively on Hadoop 2.0 and using Hadoop as a real-time streaming operating system, DataTorrent RTS offers a compelling and innovative architecture. True streaming window event processing results in a platform that can scale to billions of events, while offering inherent fault tolerance that businesses need as analytics grows in importance. Scalable, any-source data ingestion coupled with high performance low latency complex data processing lets organizations leverage their existing open big data environments and processes to deliver streams processing that is easy to program, deploy, and use.