Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

STREAMING WORKLOAD GENERATOR FOR TESTING BILLING MEDIATION PLATFORM IN TELECOM INDUSTRY

Eric Bouillet

IBM T. J. Watson Research Center
19, Skyline Drive
Hawthorne NY 10532, USA

Parijat Dube

IBM T. J. Watson Research Center
19, Skyline Drive
Hawthorne NY 10532, USA

ABSTRACT

Billing Mediation Platform (BMP) in Telco is used to process real-time streams of Call Detail Records (CDRs) which can number tens of billions a day. The comprehensive records generated by BMPs can be used for billing and accounting, fraud detection, campaign management, spam filtering, traffic analysis, and churn prediction. Many of these applications are characterized by real-time processing requiring high throughput, low-latency analysis of CDRs. Testing such BMPs has different dimensions, stress testing of analytics for scalability, correctness of analytics, what-if scenarios, all of which require CDRs with realistic volumetric and contextual properties. We propose WLG, a framework for testing and benchmarking BMPs which involves generating high volumes of CDRs representative of real-world data. The framework is flexible in its ability to express and tune the workload generation to simulate CDRs from broad range of traffic patterns while preserving different spatio-temporal correlations and content-level information observed in real-world CDRs.

1 INTRODUCTION

Typical telecom applications like billing and accounting are limited to passive analysis of data records. Internet and mobile technologies have provided new avenues for communication like Voice over IP (VoIP) calls, mobile web, and texting. This has triggered development of several novel applications built to exploit data from these communication mediums for achieving different business goals, demand generation and retention, revenue management, and data security. Examples of these emerging applications include fraud detection, campaign and promotion management including mobile advertising, spam filtering, and churn prediction. These applications employ complex analytics for processing CDR streams from ongoing calls and take responsive actions in real-time. The streaming data also has very small lifetime and the sheer magnitude of their generation (in billions per day) make the traditional store-and-process type of paradigm infeasible. Successful development and deployment of these applications prerequisite a system for very efficient ingestion, filtering and processing of call record streams. Both the hardware and software infrastructure constituting the system should be extremely fast, scalable, flexible and efficient. The development of these applications involves substantial amount of testing for analytics debugging and performance benchmarking. The current BMP infrastructure cannot support streams and hence fall short of supporting these streaming applications. There is thus a need to develop a new architecture for CDR Hubs which can meet the demands posed by emerging streaming applications and services.

Before moving an application to production we must verify the ability of the application to handle the expected rates, and ensure the correctness of its analytics. Validation is often performed by running the application in the production environment using live or pre-recorded data for a trial period of time. The main drawback of this approach is that it allows no or very limited control on the data generation, and it provides no guarantee that all the paths in the application's flow are exercised within a deterministic time interval. For instance some of the application's flow may be triggered by improbable combinations of events, or are extremely rare events, or even hypothetical events used to test what-if-scenarios that have never occurred before. Such event may not be present in the test data and would have to be inserted manually. Furthermore, the application developers may not be authorized to access real data due to privacy concerns.

The alternate approach is thus to test the application using simulated data. Simulated data can be generated randomly according to statistical properties aggregated from actual content. Such approach has the advantage that it

is less sensitive to privacy protection safeguards, and it offers more flexibility in the content generation which can be customized to validate specific analytics in the application.

At IBM Research, we have developed SWORD, a scalable and flexible workload generator for testing and benchmarking high-volume distributed stream processing systems (Anderson et al. 2006, Bouillet et al. 2008). It provides a distributed platform for generating a wide range of workload types with both volumetric and contextual correlations. The data synthesis and content modeling functionalities are implemented in different libraries in SWORD. The workload generating agents invoke application specific data factory objects (corresponding to the modality, encoding and transport level protocols) which then use the content meta-data provided by content models to generate semantically, contextually and statistically correct contents. SWORD was successfully used for testing analytics correctness and performance of multi-modal streaming applications on System S (Wu et al. 2007).

This paper introduces a new design for workload generation for stream processing systems. We have fully integrated workload generation into Infosphere Streams (IBM Infosphere Streams), or Streams, a new high performance stream processing product developed at IBM. Specifically, we have implemented a new version of SWORD, which we refer as the WLG, that takes advantage of new features in Streams, in particular its ability to support the concept of generic templatized operators. This integration of SWORD with Streams allows us to provide the WLG as a toolkit of Streams, thus fully integrated as part of the streams application, instead of a workload generator implemented on a separate platform. The WLG takes advantage of the development studio of Streams to specify schemas etc, like other operators of the streaming application, so integration of workload generation with application is straightforward. This provides two important features: (i) with the same development platform (i.e., Streams) we can develop the application and the workload generator to debug the application, and (ii) with the same runtime environment we can manage the application and the workload generator to test the application.

The paper is organized as follows. Sec. 2 describes Infosphere Streams and its programming model. Sec. 3 highlights the key components of a next generation BMP built over Streams. We also provide workload generation requirements for testing and benchmarking Streams based BMP and other applications built over it. The WLG developed using the Infosphere Streams platform for testing next generation telco hubs and applications is described in Sec. 4. We provide performance numbers for the WLG in Sec. 5. Finally we conclude in Sec. 6 with a discussion about our ongoing work.

2 INFOSPHERE STREAMS

Streams provides a massively parallel, highly efficient, scalable infrastructure supporting development, testing and execution of streaming applications characterized by very high throughput and very low latency requirements. It has been used in a variety of application domains, from environmental monitoring to algorithmic trading. It offers both a development environment and runtime support for data stream processing applications.

Streams supports structured as well as unstructured data stream processing and can be deployed to a distributed infrastructure consisting of a large number of nodes. The runtime can execute long-running jobs that take the form of data-flow graphs. A data-flow graph consists of a set of operators connected via their input and output ports by streams, where each stream carries a series of Stream Data Objects (SDOs). The operator ports as well as streams connecting them are typed, meaning that all the SDOs carried on any stream share an identical schema specific to that stream. Each operator implements data stream analytics and are distributed over the compute nodes.

SPADE (Stream Processing Application Declarative Engine language, (Gedik et al. 2008)) is the declarative stream processing engine of Streams. It is also the name of the declarative language used to program SPADE applications. Concretely, SPADE offers:

1. An intermediate language for flexible composition of parallel and distributed data flow graphs. This language sits in-between higher level programming tools and the lower level Streams programming APIs.
2. A modular, component-based programming model, which allows reuse, extensibility and rapid prototyping.
3. A toolkit of type-generic built-in stream processing operators. SPADE supports all basic stream-relational operators with rich windowing semantics.
4. The ability to extend the set of built-in operators with user-defined ones, programmable in either C++ or Java.
5. A broad range of stream adapters. These adapters are used to ingest data from outside sources and publish data to outside destinations, such as network sockets, relational and XML databases, filesystems, as well as proprietary platforms such as IBM Websphere Front Office, and IBM DB2 Data Stream Engine.

SPADE uses code generation to fuse operators into PEs. The PE code generator produces code that (1) fetches tuples from the PE input buffers and relays them to the operators within, (2) receives tuples from operators within and inserts them into the PE output buffers, and (3) for all the intra-PE connections between the operators, it fuses the outputs of operators with the inputs of downstream ones using function calls. In other words, when going from a SPADE program to the actual deployable distributed program, the logical streams may be implemented as simple function calls (for fused operators) to pointer exchanges (across PEs in the same computational node) to network communication (for

PEs sitting on different computational nodes).This code generation approach is extremely powerful because through simple recompilation one can go from a fully fused application to a fully distributed one.

2.1 Language Support via different kinds of operators

SPADE, the programming language of IBM's InfoSphere Streams, was conceived around the idea of providing toolkits of operators. These operators can be used to implement any relational query as well as windowing extensions commonly required by streaming applications. Additional convenience stream manipulation operators are also included, providing workload partitioning capabilities and generation of window boundaries, among other functionalities. We commonly refer to these operators as built-in operators (as opposed to user defined operators). The operators currently supported in the stream-relational toolkit are:

1. **Source** A Source operator is used for creating a stream of data flowing from an external source. This operator is capable of performing parsing and tuple creation as well as interacting with external devices.
2. **Sink**: A Sink operator is used for converting a stream into a flow of tuples that can be used by components that are not part of System S. Its main task consists of converting tuples into objects accessible externally through the file system or network.
3. **Functor**: A Functor operator is used for performing tuple-level manipulations such as filtering, projection, mapping, attribute creation and transformation. In these manipulations, the Functor operator can access tuples that have appeared earlier in the input stream.
4. **Aggregate**: An Aggregate operator is used for grouping and summarization of incoming tuples. This operator supports a large number of grouping mechanism and summarization functions.
5. **Join**: A Join operation is used for correlating two streams. System S can be paired up in several ways and the join predicate, i.e., the expression determining when tuples from the two streams are joined, can be arbitrarily complex.
6. **Sort**: A Sort operator is used for imposing an order on incoming tuples in a stream. The ordering algorithm can be tweaked in several ways.
7. **Barrier**: A Barrier operator is used as a synchronization point. It consumes tuples from multiple streams, outputting a tuple only when a tuple from each of the input streams has arrived.
8. **Punctor**: A Punctor operator is used for performing tuple-level manipulations, with the exception of filtering. Unlike a Functor, a Punctor can insert punctuations into the output stream based on a user supplied punctuation condition.
9. **Split**: A Split operator is used for splitting a stream into multiple output streams, based on a split condition that is used to determine which of the output streams a tuple is to be forwarded to.
10. **Delay**: A Delay operator is used for delaying a stream based on a specified amount of delay, allowing time-shifting of streams.

In addition to the existing relational algebra toolkit, the language was designed to support extensions. In particular, the developer can create User Builtin Operators (UBOPs), which are fully templatized operators. UBOPs' names and associated syntax are recognized by the SPADE language parser. The templatization makes these operators usable in a type- and stream schema- generic fashion, similarly to regular built-in operators. The UBOP support is the fundamental aspect that allows the language to provide support for the creation of toolkits geared towards other application domains like, signal processing, scientific computing, and as we will see later, workload generation.

2.2 Notation

In IBM InfoSphere Streams SPADE programming language, all operators share a common structure in terms of how they are defined. In the notation used below in figure 1, angle brackets (<>) mark the places that must be customized when employing a particular operator.

```
stream <stream_name> ( <stream_schema> )
     := MyOperator ( <input_streams> )
          [ <operator_parameters> ]
          { <output_attribute_assignments> }
```

Figure 1: SPADE operator syntax

With the exception of the Sink operator, every operator produces a stream, therefore, the way the notation above should be read is: the operator MyOperator produces the stream stream_name whose schema is stream_schema and takes as inputs the input_streams. The operator MyOperator is configured by the operator_parameters (enclosed within square brackets). The output stream attributes are assigned values based on the descriptions given by output_attribute_assignments

(enclosed within curly brackets). The output_attribute_assignments usually make reference to input attributes that are part of the input_streams.

2.3 Compiler and Runtime Support

Given an application specification in SPADE's intermediate language, the SPADE compiler generates optimized code that will run on InfoSphere Streams. SPADE's effective code generation and optimization framework enables it to fully exploit the performance and scalability of InfoSphere Streams. The reliance on code generation provides the means for the creation of highly optimized platform- and application-specific code. In contrast to traditional database query compilers, the SPADE compiler outputs code that is tailored to the application at hand as well as system-specific aspects such as: the underlying network topology, the distributed processing topology for the application (i.e., where each piece will run), and the computational environment. In most cases, applications created with SPADE are long-running queries. Hence the long running times amortize the build costs. Nevertheless, the SPADE compiler has numerous features to support incremental builds, reducing the build costs as well.

3 EXPERIMENTAL TELCO APPLICATION PILOT

Billing Mediation Platform (BMP) in Telco processes Call Detail Records (CDRs), from different telephone switches and transforms them into some standard data types, mainly for billing purposes. Current days BMPs process Extended Detail Records(xDRs) which include Event Detail Records (EDRs), IP Detail Records (IPDRs), apart from CDRs, from different network elements (like telephone, cable and DSL switches). The comprehensive records generated by BMPs can be used in different applications like mediation, billing, fraud detection, campaign and promotion management, spam filtering, traffic analysis, churn prediction and prevention, and capacity planning. Many of these applications are characterized by real-time processing and prediction requiring high throughput, low-latency analysis of xDRs. Thus it is crucial to have efficient BMP architecture for real-time ingest, processing and archiving of xDRs. With the influx of mobile technology and Internet telephony there is an unprecedented surge in the xDRs which can typically range from billions to tens of billions a day for large telecom operators. There is also an increase in demand for "real-time" services requiring high throughput and low latency analysis of xDRs.

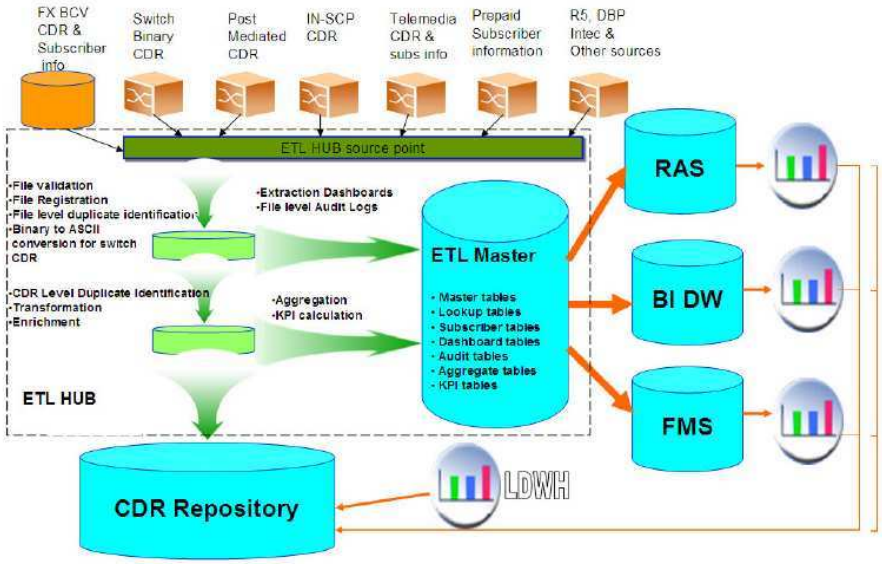

Figure 2: Functional diagram of an xDR Hub

We are building next generation xDR Hubs based on the InfoSphere Streams platform. Many applications are being developed that take responsive actions in real-time based on call patterns and customer behavior, e.g., campaign management, fraud detection (Xin and Girolami 2007), and churn-prediction (Wei and Chiu 2002). These applications employ very efficient analytics working on real-time CDRs. Hence the xDR Hub needs to have architecture and middleware supporting streaming data processing. Figure 2 is a functional diagram of an InfoSphere Streams xDR Hub. There are several requirements from the xDR Hub:

- **Efficient Ingestion of xDRs of different types, formats and volumes:** As shown in Fig. 2 the traffic comes to xDR Hub from different sources and applications. There are subscriber information CDRs, switch CDRs, post-mediated CDRs, FX CDRs etc. coming from different types of switches and at different rates at the Hub. xDRs can correspond to different modalities like voice data (VoIP, telephone, mobile), text data (SMS) or multimedia data (MMS).
- **Registration and Error Handling:** Typically xDRs are grouped into files at the xDR Hub rather than individual xDRs. There is a possibility of duplication of files due to switch restarts. At record registration, the xDR Hub needs to check for duplicate files and de-duplicate them. There can be several combinations of attributes that can be checked for file duplication like number of data records in the file, time of generation of file/CDRs, switch where the file was generated, to name a few.
- **Rule Based Data Transformations:** For each CDR several table look-ups need to be performed to identify different attributes of the record that are not explicitly in the CDR and join the CDR attributes with these at-rest data attributes (varying at slower time scale compared to CDRs) like subscribers info, billing plans, prefix numbers, account value prior to call registration. The at-rest data can itself range in millions. Thus a xDR Hub needs to look-up phone number prefix (area-code), subscriber info like calling plan, call type (local, long-distance, international), special services call (toll free, public services, intra-organization) and join the resulting at-rest attributes with CDR attributes. The extended set of attributes is then used to generate unique keys for each CDR where a key can be a concatenation of several attributes of the CDR. The reformatted (with extended attribute sets) CDRs are then useful for applications dealing with billing, promotion and subscriber KPIs calculations.
- **CDR De-Duplication:** The xDR Hub performs individual CDR level error checking and de-duplication.
- **Aggregate Computation:** KPIs (Key Performance Indicators) for different subscribers are evaluated at the Hub. KPIs include volumetric data based on geographic location, social connections, and customer specific attributes like age, gender etc. These KPIs are used by different applications like subscriber revenue analysis and forecasting, promotion management and churn prediction.

Figure 3 shows different steps involved in the processing of xDR streams within the InfoSphere Streams xDR Hub.

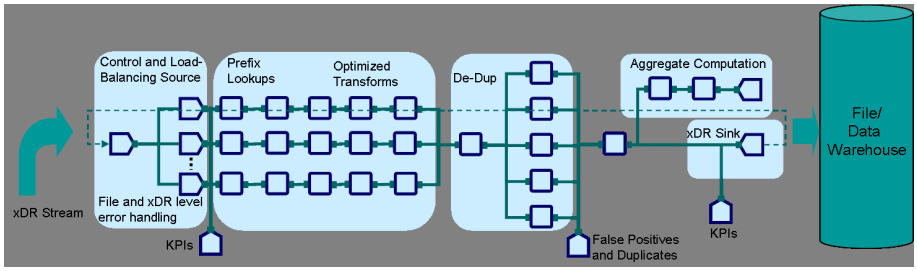

Figure 3: Processing of xDR Streams at the InfoSphere Streams xDR Hub

Testing and benchmarking of xDR Hubs has different dimensions, stress testing of analytics for scalability, correctness of analytics to identify duplicate false-positives, what-if test scenarios, all of which requires xDRs with realistic volumetric and contextual properties. A workload generator for testing and benchmarking xDR Hubs has several requirements:

- **Diversity of Volume, Modality, Format:** A typical large xDR Hub is predicted to receive billions of xDRs per day. Thus the workload generator should have the ability to generate very high volumes of xDRs (10 Billions/day = 100 K/sec). Further generated workload should have a mix of xDRs of different modalities (voice, text, multimedia) and formats (plain ascii, or ASN.1 binary encoded).
- **Erroneous and Duplicate Records:** For testing components handling errors and performing de-duplication the workload generator needs to generate records having erroneous fields and/or field values, duplicate files and duplicate xDRs. The de-duplication and error handling components look at different attributes of files and xDRs. The generator should be able to exhaustively generate different possible cases of duplication and errors. There is also a dependency between attributes of records which should be maintained during workload synthesis so as to avoid trivial detection of errors.
- **Spatio-Temporal Correlation:** The social sphere of interaction of individuals generate different types of dependencies in their calling patterns and hence the generated xDRs:
  - *Spatial Correlations:* xDRs corresponding to calls made by individuals to their home, to their friends' home, to work place of family members and friends.

  - *Temporal Correlations:* xDRs corresponding to calls made by individuals at different times of day e.g., day time calls to business partners in same geographic locations, night time calls for business partners in different time zone.

  The workload generator should be able to generate xDRs capturing spatio-temporal correlations defined by different social networks like family, friends and work.
- **Controllable and Reproducible Generation:** xDRs conforming to volumetric and statistical attributes of current and future demand seen from different subscribers to test KPI generation analytics which are used by subscriber-centric applications like advertising and revenue management.
- **Exhaustive Generation:** The generated data records must cover all the possible execution paths that can be taken by a record in the xDR Hub. This includes generating records, (i) having different types of errors and duplication at file and CDR levels, (ii) exercising all the different look-up rules to test correctness and scalability of appplications like business intelligence analytics.

In the next section, we develop a stream based workload generator which synthesizes high volumes of xDR streams representative of real-world data seen by an xDR Hub. The simulated xDRs preserve spatio-temporal correlations and social-network dependency observed in real-world xDRs. The framework is flexible in its ability to express and tune the workload generation to simulate xDRs from broad range of traffic patterns while preserving different spatio-temporal correlations and content-level information. The framework also has the ability to model application-level data which can be used to generate traffic representative of future applications or services for market data analysis.

4 SYNTHETIC WORKLOAD

We have implemented an application testing framework that satisfies the requirements stated in the previous section. The framework consists of an InfoSphere Streams toolkit of UBOP specialized for the workload generation. Our design permits a seamless integration between the workload generation and the tested application. Tested applications may contain one or more such operators, each of which produces a data stream generated according to statistical properties specified in a *Content Model Description* document. Additionally, our WLG operators accept optional input streams that receive user- or application-generated commands for controlling the content generation. WLG operators can also periodically send data rates, CPU usages, and other information about their states on monitoring streams. Monitoring streams can be connected to a *WLG Controller* operator, another component of our WLG application framework. *WLG Controllers* aggregate the information received from connected WLG operators, forward the aggregated information to remote clients, and send client generated commands back to connected WLG operators. The complete architecture is illustrated in Figure 4.

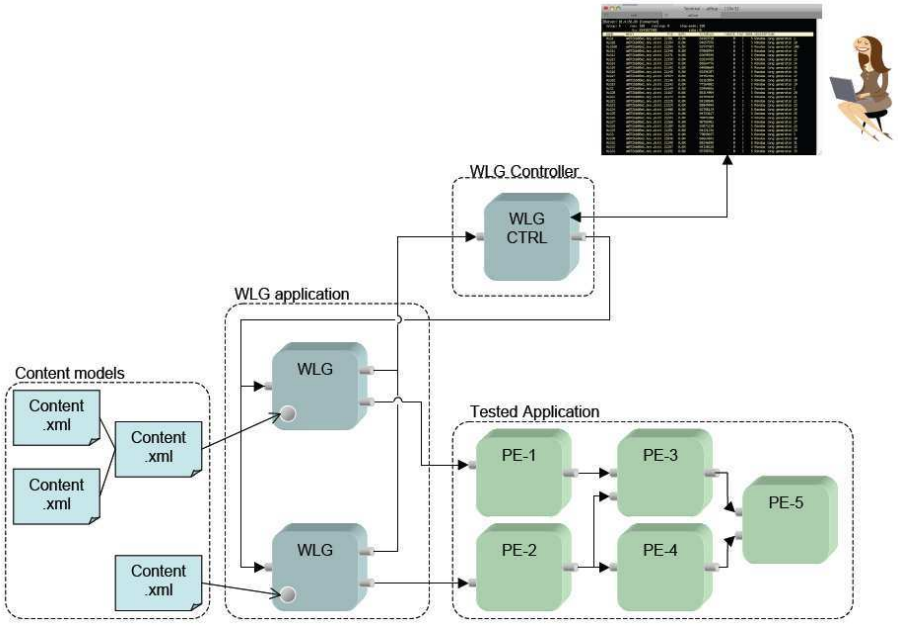

Figure 4: Workload Generator Architecture

In the following we review in details the various components of the WLG for data stream application framework, namely *WLG operators*, *Content Model Descriptions*, and *WLG Controllers*.

4.1 WLG Operator

WLG operators are implemented as UBOP and are thus declared using the same notation as any other builtin-operators in the SPADE language as described earlier in section 2.2. The example in figure 5 shows such a declaration in SPADE.

```
stream cdrData(IMEI : String, IMSI : String, MSCID : String, MSISDN : String, MSRN : String,
  DateTime : String, callType : Integer, dateOfCall : String, durationOfCall : String,
  calledNumber : String, callingNumber : String, prepaidFlag : Boolean, timeOfCall : String,
  usageType : String)
    := wlg()[
        model : "telcos.cmdl#cdrs"
    ]{}
```

Figure 5: A simple WLG application in SPADE

Note that the schema of the operator output stream *cdrData* is application specific (for illustration purpose we only show a subset of the CDR fields used in the actual application). This is possible because WLG is a fully templatized operator. At compile time SPADE automatically converts the templates into application specific C++ code. This approach allows the creation of very generic UBOPs that support variable number of input and output ports, schema, and behaviors.

With the WLG operator it is possible to express the properties of the content, such as ranges of allowed values for individual fields, correlations between fields, and data rates. The content and its statistical properties is specified in a content model document, which is described next. The URL of the content model is provided as a configuration parameter to the operator (the model parameter in the example above). The operator compiles the document at runtime into an abstract syntax tree and executes the syntax tree in a continuous loop. SDOs of the WLG operator's output streams and their fields are generated during the execution of the syntax tree. If desired the same WLG operator can be instructed to recompile the content model document dynamically with a new content and statistical properties specifications. This design is particularly useful for nowcasting prediction applications based on simulation models. Such applications comprise a training process which generates new models from observation data, and a simulation process which generates predicted outcomes using those models.

4.2 Content Model Description Language

In the WLG we improved the concept of content models presented in (Anderson et al. 2006, Bouillet et al. 2008). Like their earlier versions, content models are expressed in the form of a decision flow-graph. The advantages of the decision flow-graph representation, is that unlike most other programming languages, they are simple to understand and use. They also require little data preparation, for instance they can easily be created directly from the output of decision tree learning algorithms used in data mining or machine learning. When the WLG generates a new SDO, it starts from an initial position of the graph, and follows a random path according to decision logics implemented in interior nodes traversed by the random path. During the traversal some of the nodes, notably leaf nodes, may operate on the fields of the SDO being generated, such as setting the value of the field. After executing a node's operation the WLG returns to the parent node, and repeats this process until it returns to the starting node, at which point it sends the SDO constructed during the traversal and starts a new iteration.

Example of decision logics include:

- Traverse a randomly selected adjacent node (selection may be conditional to previously traversed nodes).
- Traverse all adjacent nodes.
- Traverse adjacent nodes in a round robin fashion.
- Repeat an adjacent node a number of time.
- Specify a starting node for the next SDO generating iteration.

Other decision flow-graph nodes operate on the fields of the generated SDO. Supported types of operations are:

- Operations that set the *Left Hand Side* (LHS) for the next value assignment. The LHS is typically a field of the SDO, or an internal state used in the content model's decision logics. The LHS remains in effect until a new LHS is set, or until the random path backtracks to that node. When backtracking, the LHS is reset to its previous value.

- Operations that set or append a value to the current LHS. The value can be a constant (string, or numerical), a function of the current time, or it can be randomly generated. Several distribution functions are available for generating random values, such as uniform, normal, exponential, and random walks.
- Transform operations, e.g. convert a timestamp into human-readable format.
- IO operations, e.g. read next value from file, used to replay pre-recorded content.

Another class of flow-graph nodes provides functions for controlling the behavior of the WLG. Operations performed by those nodes are:

- Wait for a predetermined or random amount of time.
- Wait until it receives further instruction to resume the random walk; used as breakpoints.
- Increment a performance counter: InfoSphere Streams provides a system level API allowing applications to create and operate on named performance counters. An interface is also provided to visualize performance counters (cumulated or delta values) as time series.

Breakpoints and performance counters are two important features of the WLG. They are used to pause or flag critical sections of the decision graph. Those sections may consist of challenges used to validate the analytics of the application. For instance a performance counter which is incremented when the WLG generates a duplicate CDR can be tallied with duplicates detected by the application. With breakpoints the WLG can be paused before or after sending the duplicate CDR, giving the user an opportunity to inspect the application while it is processing the CDR.

The content model are written in XML. Figure 6 shows a simplified version of the content model:

```
<cmdl xmlns="http://www.ibm.com/distillery/atf/cmdl/1.0 ">
  <forall id="cdrs">
    <set name="dateOfCall"><ctime format="%Y%m%d"/></set>
    <set name="timeOfCall"><ctime format="%H%M%S"/></set>
    <pick>
      <c w="45">
      <forall>
        <set name="callingNumber"><s>+17778888</s></set>
        <set name="calledNumber"><ref name="contacts1"/></set>
      </forall>
    </c>
    <c w="45">
      <forall>
        <set name="callingNumber"><s>+17778889</s></set>
        <set name="calledNumber"><ref name="contacts2"/></set>
      </forall>
    </c>
    <c w="10">
      <forall>
        <set name="callingNumber"><ref name="randNumber"/></set>
        <set name="calledNumber"><ref name="randNumber"/></set>
      </forall>
    </c>
  </pick>
</forall>
<pick id="contacts1">
    <c w="50"><s>+17778889</s></c>
    <c w="45"><s>+17778001</s></c>
    <c w="5"><ref name="randNumber"/>/c>
</pick>
<pick id="contacts2">
    <c w="60"><s>+17778888</s></c>
    <c w="35"><s>+17778001</s></c>
    <c w="5"><ref name="randNumber"/></c>
</pick>
<cat id="randNumber"><s>+1</s><rand min="1000000" max="9999999"/></cat>
</cmdl>
```

Figure 6: Sample of Content Model Description Language (CMDL)

Content models are organized into tree structures, but nodes have an optional *id* attribute through which other nodes (e.g. *ref* nodes) can link to in a more arbitrary fashion, therefore the resulting graph is not limited to a tree representation. The above content model describes the calling patterns between two distinct callers and callees from their respective contact lists. This pattern is mixed with ten percent of randomly generated callee and caller phone numbers. This example is for illustration purposes. In our experiments we used a million callers with an average list of ten contacts each for a total memory footprint of 1.6Gbytes. Even larger content models can be decomposed into

multiple WLG operators distributed on several computers. The content model sets the values for the *dateOfCall* and *timeOfCall* fields of the output SDO using the current time with a user specified format (*ctime*). It then selects a random value pair (*pick*) for the *callingNumber* and *calledNumber* fields. The two fields are set simultaneously within a *forall* node to express their social relationships. Callee numbers are selected from separate trees, *contacts1* and *contacts2*. Contact lists are implemented using *pick* decision logics, which randomly select child decision logic nodes according to user specified weights. As shown in the example, the contact numbers selected by the pick decision logics can be either a known contact, or with five percent probability it can be a random number. Nodes of type *set* bind values assigned by subsequent nodes to a particular field of the SDO, identified by the *name* attribute. In order to improve reusability of the content models, we ignore bindings to non-existent SDO fields.

4.3 WLG Controller

As illustrated in Figure 4, WLG operators can have output streams for reporting their current state, and input streams for controlling the content generation from other parts of the application. Our application test framework includes a WLG Controller UBOP to enable a human interaction in the control loop. The WLG Controller listens to one or more WLG operator monitoring streams, summarizes the collected information, and streams the summarized information to remote client interfaces connected to the WLG Controller on its TCP service port. The interface provides the users with a choice of commands, such as pause, resume, stepping, increase or decrease SDO rate. WLG operators can have multiple control and monitoring streams, possibly with different stream schemas. WLG Controllers broadcast user commands back to their managed WLG operators. A command can affect a single WLG or a group of WLG operators. The assignments of operator into groups is a WLG operator configuration parameter determined at compilation time in the SPADE program. Commands broadcasted to the WLG operators contain the group ID of the targeted operators so that other operators not in the group can ignore it. Using this approach we can control a hundred operators with very little overhead. The monitoring information is sent by default at one second intervals (per WLG operator). The rate increases when an operator receives a control SDO, because it immediately responds with a monitoring SDO for control feedback. Even so, the rate of monitoring SDO remains several orders of magnitude less than typical rates observed in our data stream processing applications.

The SPADE program in Figure 7 shows an example of application using a single WLG attached to a WLG controller. In this example stream *wlgMonitoring* contains fields reporting the state of the WLG operator. Fields *wlgTupleRate* and *wlgTupleTx* indicates the SDO rate and total number of SDOs generated by the operator; fields *wlgCPU* reports CPU usage; fields *wlgMode* indicates if the operator is in step mode or in continuous mode; fields *wlgStatus* reports its status (e.g. idle, running, terminating). Other fields are descriptive fields, such as WLG description, host where the WLG is running, and current content model document. The control stream includes field *wlgGid*, which specifies the group ID of operators that should honor the command. The list of groups that an operator belongs to is indicated by the *gid* configuration parameter. Other fields used in the example are *wlgMode*, which specifies that the WLG should operate either step-by-step, or continuously, and *wlgNumTuple*, the number of SDOs sent in one step. Other commands not shown here are available, such as increase, decrease the rate, or change the content model. Command fields can be bundled in the same control stream or separated into separate streams; in either case SPADE generates the code appropriately from the UBOP template.

```
stream cdrData(IMEI : String, IMSI : String, MSCID : String, MSISDN : String, MSRN : String,
  DateTime : String, callType : Integer, dateOfCall : String, durationOfCall : String,
  calledNumber : String, callingNumber : String, prepaidFlag : Boolean, timeOfCall : String,
  usageType : String)
stream wlgMonitoring(wlgGUID : Long, wlgHostIp : String, wlgHostName : String, wlgCPU : Float,
  wlgPID : Integer, wlgCMDL : String, wlgGroups : LongList, wlgDescription : String,
  wlgLabel : String, wlgTupleTx : Long, wlgTupleRate : Double, wlgStatus : Integer,
  wlgMode : Boolean )
    := wlg(wlgControl)[
        gid : [1, 2, 3 ] ;
        seed : 12345 ;
        model : "telcos.cmdl#cdrs"
    ]{}

stream WlgCtrl(wlgGid : Long, wlgMode : Boolean, wlgNumTuple : Long)
    := wlgsrv(wlgMonitoring)[
      URL:"stcp://localhost:9500/"
    ]{}
```

Figure 7: Example of SPADE WLG Operator with WLG Controller

The data stream application programming model allows us to manage an arbitrarily large number of WLG operators in order to achieve very high throughputs. Managed WLG operators can have distinct data stream schemas, generated

from different description content models, and each can be individually controlled in order to fine tune the mix of different types of data streams.

5 PERFORMANCE MEASUREMENTS

We tested the performances of the WLG on a single server comprising four Intel Xeon 5160 3Ghz cores and eight gigabytes of RAM. We used two content models: (1) an extended version of the sample content model presented in Listing 6, consisting of 1 million callers with a contact list of ten entries each, and (2) the content model used to test the xDR Hub application and derived from real data, with 37 CDR fields - 8 numerical fields with various random distributions, 29 strings, each selected at random from variable lengths list ranging from 2 to 20000 predefined values, and complex cross-field statistical dependencies.

Using the simpler content model we achieve a rate of 320 thousands CDR/s. The memory requirement for this content model is about 1.6Gbytes. Note that callers and callees are each selected according to user-specified random distributions which consume more CPU resources to generate. If instead we replace all the distributions with uniform distributions, the total rate increases to 400 thousands CDR/s per operator.

Using the content model created for our xDR Hub application, we achieve a rate of 60 thousands CDR/s per operator. We expected the rate to be less than the rate obtained in the previous experiment, because it is characterized by a higher number of fields and more complex statistical properties. Another reason is that the CDR size is significantly larger with 300bytes on the average, and it thus costs more to transmit the CDRs over the network. If we exclude the cost of serialization, deserialization and network transmission, the rate is 80 thousands CDR/s.

In order to achieve larger rates it suffices to deploy several WLG operators in parallel, possibly on several computers (Bouillet et al. 2008). Our observations indicate that the above CDR rates increase linearly with the number of operators and stabilizes when the pool of CPU cores made available to Streams' runtime environment is 100% utilized.

6 CONCLUSIONS

Telecom applications and services are expanding to real-time sense and respond model thus requiring quick processing of streams of communication data records arriving at very high rates. The xDR Hub designed and being developed over Streams platform meets the requirements posed by the emerging stream-based telecom applications. The WLG, integrated with Stream, enables debugging and testing of streaming applications in the development environment over a common platform. The modular and extensible design of the WLG establishes itself as the top choice for a low-overhead, scalable, flexible and controllable testing and benchmarking tool for a myriad of stream-based telecom applications achieving different business goals of telecom service providers. The usability of the current WLG can be further improved with functionalities to ease in the creation of content model description documents. Example of desirable functionalities comprises tools that automatically create content model from a corpus of real data, tools that convert the output of existing data mining products into content model, and authoring tools to edit the content models. The inclusion of such functionalities would prove a powerful addition to our test framework, and are being investigated as future work.

ACKNOWLEDGEMENTS

The authors would like to thank Henrique Andrade, Hanhua Feng, Bugra Gedik, Nagui Halim, Vibhore Kumar, Dimitrios Pendarakis, Anand Ranganathan, Anton Riabov, Deepak Turaga, Octavian Udrea and Olivier Verscheure for several useful discussions during the course of this work.

REFERENCES

Anderson, K., J. P. Bigus, E. Bouillet, P. Dube, N. Halim, Z. Liu, and D. Pendarakis. 2006. Sword: Scalable and flexible workload generator for distributed stream processing systems. In *Proc. of 2006 Winter Simulation Conference*, ed. L. F. Perrone, B. G. Lawson, J. Liu, and F. P. Wieland, 2109–2116. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Bouillet, E., P. Dube, D. George, Z. Liu, D. Pendarakis, and L. Zhang. 2008. Distributed multi-layered workload synthesis for testing stream processing systems. In *WSC '08: Proceedings of the 40th Conference on Winter Simulation*, 1003–1011: Winter Simulation Conference.

Gedik, B., H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo. 2008. SPADE: the System S declarative stream processing engine. In *SIGMOD 2008*, 1123–1134.

IBM Infosphere Streams. http://www-01.ibm.com/software/data/infosphere/streams/.

Wei, C.-P., and I.-T. Chiu. 2002. Turning telecommunications call details to churn prediction: a data mining approach. In *Expert Systems with Applications*, 103–112.

Wu, K.-L., P. S. Yu, B. Gedik, K. Hildrum, C. Aggarwal, E. Bouillet, W. Fan, D. George, X. Gu, G. Luo, and H. Wang. 2007. Challenges and experience in prototyping a multi-modal stream analytic and monitoring application on system s. In *Proc. of Very Large Database Systems (VLDB)*, 1185–1196.

Xin, D., and M. Girolami. 2007. Employing latent dirichlet allocation for fraud detection in telecommunications. In *Pattern Recognition Letters*, 1727–1734.

AUTHOR BIOGRAPHIES

ERIC BOUILLET is a Research Staff Member at IBM T. J. Watson Research Center, NY, where he works on data modeling and test data generation. Before joining IBM, Dr. Bouillet has worked at Tellium Inc. from 2000-2004 and at Bell Labs/Lucent Technologies from 1998-2000. Eric holds an M.S. and a Ph.D. in electrical engineering from Columbia University. He also holds a joint degree from l'Ecole Nationale Suprieure des Telecommunications (ENST) Paris and EURECOM Sophia Antipolis. His current research interests include data modeling and test data generation, design of optical networks and optimization of lightpath provisioning and fault restoration algorithms. His email address is <ericbou@us.ibm.com>.

PARIJAT DUBE received his M.S. in Electrical Communication Engg. from Indian Institute of Science, Bangalore in 2001 and his Ph.D. in Computer Science from University of Nice-Sophia Antipolis in 2002 where he was affiliated to INRIA. He joined IBM T. J. Watson Research Center, Hawthorne, New York in 2002. His research interests include performance analysis and control of computer systems, distributed computing, stochastic modeling and game theory. His email address is <pdube@us.ibm.com>.