Towards Efficient Dissemination and Filtering of XML Data Streams

Kirill Belyaev
Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873
Email: kirill@cs.colostate.edu

Indrakshi Ray
Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873
Email: iray@cs.colostate.edu

Abstract—**The vast amounts of data generated in near real-time due to prolific use of sensors, pervasive usage of mobile Internet, and popularity of social media platforms, necessitates the efficient dissemination of the semi-structured streaming data to the consuming applications. Towards this end, we introduce the subscriber-centric XML filtering approach for seamless and efficient XML stream replication/distribution mechanism. The subscriber-centric filtering architecture can be configured to support different topologies in order to support efficient message filtering for a large number of concurrent subscribers. It allows selective filtering on the various nodes that improves efficiency and provides applications with data on a need-to-know basis. Moreover, it supports interoperability and allows semi-structured streams generated from multiple sources to be filtered. Our XML filtering network consists of decoupled data producers, message transformation agents and XML brokers that can be deployed in conventional data centers as well as in the public cloud environment. We provide detailed performance results of processing filtering queries in several use case scenarios with varying XML message loads and number of nodes involved in the replication/dissemination process. Our results indicate that the subscriber-centric XML filtering architecture is a viable approach for disseminating semi-structured data streams to the various consuming applications.**

I. Introduction

With the increase in the usage of mobile devices that are connected to the Internet, consumers are subscribing to various types of applications, such as, Yahoo! Weather, Yahoo! Finance, and Twitter, that require delivery of streaming data in a timely manner. There is a need for gathering semi-structured streaming data from the sources, transforming them to a form that facilitates interoperability, and then replicating/distributing the data stream in an efficient manner to the multifarious applications needing the data for various purposes, such as forwarding the data to the subscribing consumers and/or performing complex stream analytics to detect trends or outliers.

Publish/subscribe (pub/sub) has been a popular communication paradigm which provides customized notifications to users in a distributed environment [1]. Pub/sub systems are used in geomarketing, traffic and weather alerts, emergency response services, and social networking. These systems are large (e.g. Twitter is estimated to handle over 400 million tweets daily), geographically distributed and largely subscription based. Subscriptions in such systems, such as deals for local shops and traffic alerts for freeways, involve simple queries and are short-lived. Such pub/sub systems provide very little query support and trade expressiveness for performance. However, their inability to express expressive continuous queries over data streams, possibly in different formats, make them unsuitable for detecting complex events that arise in situation monitoring applications.

The majority of modern Internet applications use XML as an inter-application communication exchange format in spite of its heavy network bandwidth utilization. Typically, the applications generate data in XML so that it can be easily distributed to other applications by operational runtime environments [2] [3] [4]. XML-based data dissemination networks are starting to become a reality [4]. Data generated in XML format should be adapted for efficient streaming, filtering and consumption by the subscribing applications. We address this issue by introducing the *subscriber-centric* XML content filtering service where each XML message generated or received by the application layer is transformed into a dissemination-ready XML message for transport over the network infrastructure.

We propose the TeleScope XML filtering broker [5] in this paper to carry out the selective dissemination/replication of XML messages to consuming end-points. Our subscriber-centric broker has the following characteristics. (i) Fast processing of XML messages under high input stream rates and large number of subscribers – the TeleScope XML filtering broker is written in C that supports very fast message filtering speeds even with a large number of concurrent subscribers. (ii) Content-based XML filtering uses expressive filtering language – TeleScope introduces an engine with simple yet efficient user-friendly content filtering domain specific language parser over XML stream with full support for Boolean logic operators as well as supplemental operators such as network prefix range computing operators. The language allows easy integration with XML consuming applications and does not require the knowledge of complex XPath/XQuery [6] semantics, but supports the common stream filtering/dissemination scenarios. (iii) Ability to form the overlay filtering network for XML dissemination – placing of TeleScope nodes in the form of the filtering mesh allows efficient dissemination of XML content to the endpoints.

The rest of the paper is organized as follows. Section II gives a detailed overview of the XML stream replication for consuming applications and describes our subscriber-centric filtering architecture. Section III highlights the XML filtering framework. Section IV describes the subscribers management involved in the task of efficient stream dissemination. Section

V demonstrates the feasibility of our approach by describing the prototype and presenting our experimental results. Section VI provides discussion on the architectural properties of the proposed service with potential improvements. Section VII lists some relevant literature. Section VIII concludes the paper.

II. Subscriber-Centric Content Filtering Service Architecture

The majority of data generated by modern application services is in XML format and is streamed to the subscribing applications via the network. Like [7] and [8], we focus on the processing of relatively small (1 KB to 256 KB) XML messages, typical in many Internet applications, that are arriving at very rapid rate which must be delivered to large number of connected subscribers. The streaming source data must be efficiently delivered to end point applications which get only the relevant messages. A single broker instance using content-based pub/sub model may not have sufficient CPU resources to efficiently process multiple filtering queries and also support a large number of concurrent subscribers [9].

We propose a *subscriber-centric* XML content filtering service where each XML message generated or received by the application layer undergoes a simple transformation where the XML attribute length is added to the opening XML tag of the message and this transformed message is efficiently disseminated over the network infrastructure. The incremental message transformation provides a broker with the position of individual XML message in the data stream so that its network processing component is able to read incoming messages correctly at their exact boundaries and avoid XML corruption due to under-reading or over-reading into the (possible) next message in the stream.

Our subscriber-centric filtering architecture is hierarchical, where the child brokers subscribe to the parent broker and distinct operations are done by the parent and child brokers. In our model, content-filtering is done at the child brokers and the parent brokers are responsible for the efficient dissemination of messages to the child brokers. The publisher or the root node no longer has to perform tasks such as query registration, maintenance of the state of the multiple filtering queries and the association of queries to subscribers [10], [1].

Each TeleScope [5] instance is capable of operating in dual-mode being either a publisher or a subscriber or incorporating both modes of operation at the same time. The parent broker focuses on processing high rate of incoming XML messages and serving them to a large number of concurrent subscriber instances. The parent broker enforces fair queuing of XML messages to all subscribers through allocation of a separate message queue for each subscriber connection. The child broker registers the content filtering query on the incoming stream and performs filtering without consuming the computational resources of the parent broker.

Since the majority of application web services exchange messages on top of the HTTP/HTTPS, our architecture requires the application layer to connect to the stream source and request the XML stream that is then transformed into a dissemination-ready XML format. The messages are then loaded into the TeleScope broker in-memory queues and served to the subscribers. Each subscribing TeleScope broker connects

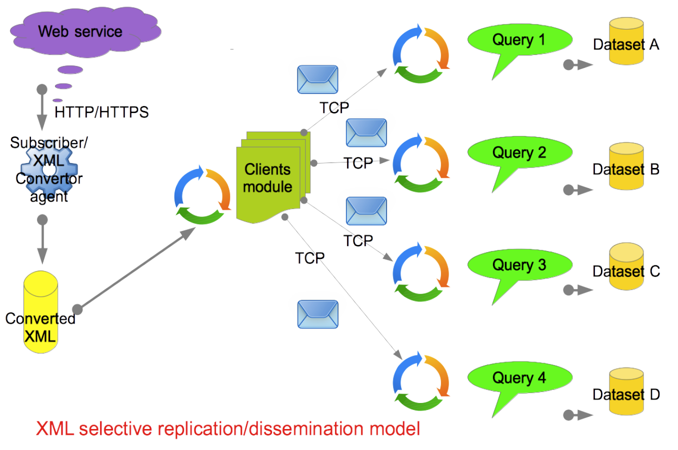

Fig. 1. Selective Replication/Dissemination Architecture

to the publishing broker through direct TCP/IP connection by specifying the IP address and the listening port of the upstream broker. In short, we employ the pull-based content delivery mechanism [1] to receive the XML stream from the upstream publisher.

Selective replication/dissemination architecture is depicted in Figure 1. The selective replication is carried out by the subscribing broker instances that receive the XML stream from the parent broker and register the distinct content filtering queries at their end for a subset of the stream. The parent broker reads XML messages from the filesystem storage that is written by the subscribing agent that is subscribed to the web service stream and performs the transformation of received XML messages into the TeleScope compliant XML format.

In the above scenarios the XML distribution is organized through forming a filtering mesh network (overlay) that starts from the root (parent) broker that receives the XML message stream either from a network connection (acting as a subscriber) or directly from the filesystem storage and continues to the leaf subscriber nodes that in turn can form the distribution meshes of their own depending on the service requirements [3]. A sample filtering overlay operating on stock exchange semi-structured data stream is depicted in Figure 2.

III. XML Filtering Engine Description

Current TeleScope filtering engine supports basic XML filtering based on elements and attributes within the XML message. The engine uses the LibXML2 [11] based XML C parser library to parse the XML message into a tree using the DOM [12] approach. We do not use the underlying XPath or XQuery facilities for the purpose of content filtering as we focus on providing user-friendly content filtering semantics [13] having the adequate expressive power.

In our current work we focus on real-time filtering of every XML message in the stream and therefore use an alternative approach without reliance on XPath framework. We have developed the Domain Specific Language parser for the filtering engine within TeleScope instance that operates on

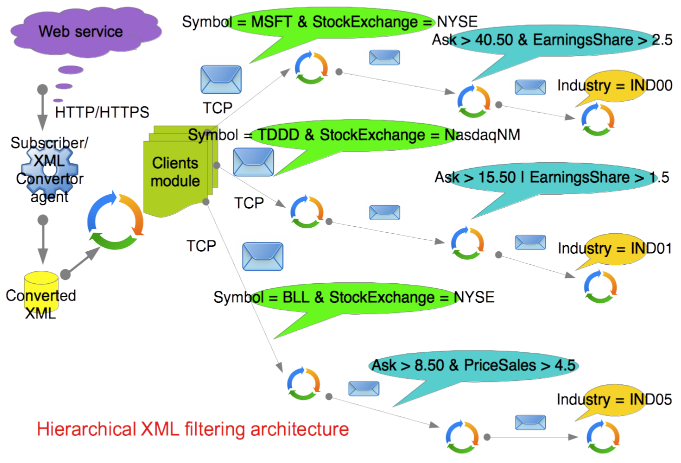
Web service
HTTP/HTTPS
Subscriber/ XML Convertor agent
Converted XML
Clients module
TCP
TCP
TCP
Symbol = MSFT & StockExchange = NYSE
Ask > 40.50 & EarningsShare > 2.5
Industry = IND00
Symbol = TDDD & StockExchange = NasdaqNM
Ask > 15.50 | EarningsShare > 1.5
Industry = IND01
Symbol = BLL & StockExchange = NYSE
Ask > 8.50 & PriceSales > 4.5
Industry = IND05
Hierarchical XML filtering architecture

Fig. 2. Overlay Filtering Architecture

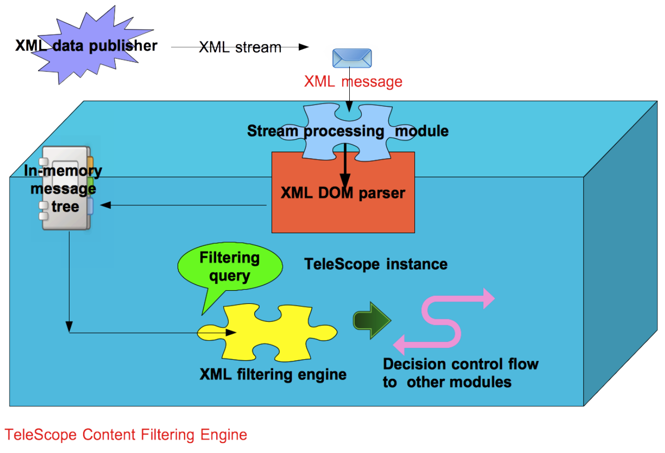

Fig. 3. Filtering Engine

the nodes of the XML message parsed into a DOM tree. The process is depicted in Figure 3.

We begin by describing the operators currently supported by our TeleScope filtering engine in Table I with sample use cases.

TABLE I. TELESCOPE FILTERING ENGINE OPERATORS

Operator	Description
=	equality operator (Symbol = ABTE)
!	not-equal operator (Symbol ! TDDD)
<	relational less than operator (CurrentPrice < 116.0)
>	relational greater than operator (CurrentPrice > 116.0)
&	logical AND (Industry = IND03 & Year = 2002)
|	logical OR (Symbol = TDDD | Symbol = ACMR)
%	substring match operator (Industry % IND)

TeleScope filtering engine provides a set of operators, shown in Table II, designed specifically for processing network prefixes including CIDR ranges. The operators follow the PREFIX element with the subsequent network range value. These are ordinary English letters that have special meaning when used within the expression.

Parentheses () are used in complex queries to separate out the component queries. Parentheses are chained through logical OR operator. Equality and negation operators (= and !) can be used in expressions involving both string and integer values where the evaluation depends on the type of operands.

The filtering query could be formulated, changed and reset via the command line shell interface through a connection to the service port (if the instance is running in the server mode). The filtering query could be considered either: (i) atomic – involving filtering condition on one attribute and having one type of logical operator (ii) composite – involving filtering condition on multiple attributes and having multiple types of logical operators The composite filtering query could be constructed out of a number (two and more) of atomic filtering queries connected via a logical OR operator and a set of parentheses. Therefore our language parser is in disjunctive normal form (DNF) that is a standardization (or normalization) of a logical formula (in our case the filtering query expression) which is a disjunction of conjunctive clauses; otherwise put, it is an OR of ANDs also known as a sum of products.

Samples of atomic and composite filtering queries based on the NASDAQ stock quotes trading are shown in Tables III and IV respectively.

IV. MANAGING SUBSCRIBERS

Efficient management of concurrent subscribers lies at the heart of successful content filtering service. Each message in the XML stream received at the front-end broker must be disseminated to every instance subscriber without delays and loss. This property guarantees that each and every subscriber has identical copies of the stream elements in the scenario of per-element stream replication. The architecture is depicted in Figure 4.

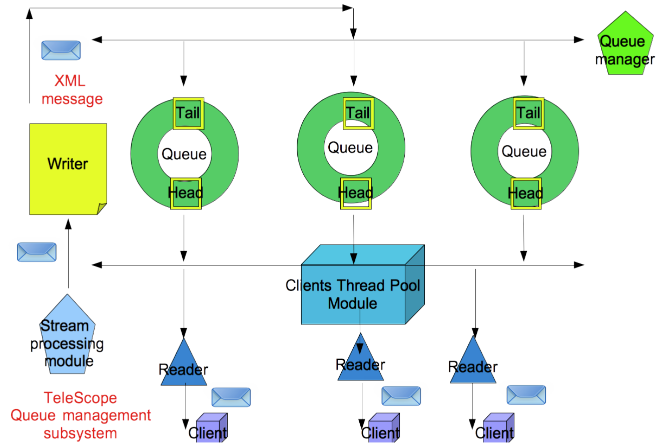

Fig. 4. Queue Management Subsystem

TeleScope subscriber-centric content filtering architecture has its own approach to address the fair subscriber management requirements. It is facilitated through fair queuing of XML messages to all subscribers by means of allocation of a separate message queue for each subscriber by having a single writer thread in TeleScope that writes every XML stream message to all the subscriber queues. This is advantageous over a single queue model where multiple subscribers access the

TABLE II. PREFIX OPERATORS

Operator	Description	Example use	Semantics
e	exact prefix match operator	PREFIX e 211.64.0.0/8	matching the networks with the exactly defined network prefix range.
l	less specific prefix match operator	PREFIX l 211.64.0.0/8	matching the networks with less specific network prefix range.
m	more specific prefix match operator	PREFIX m 211.64.0.0/8	matching the networks with more specific network prefix range.

TABLE III. ATOMIC FILTERING QUERIES

Q1: Symbol = TDDD
Q2: Sector = SEC0
Q3: Symbol = TDDD & Industry = IND00
Q4: Industry = IND00 & Year = 1999
Q5: Symbol = TDDD | Symbol = ACMR
Q6: Sector = SEC0 | Sector = SEC2
Q7: Industry = IND00 | Industry = IND01 | Industry = IND02
Q8: CurrentPrice < 116.0 | OpenPrice > 177.0 | Change = -0.52

TABLE IV. COMPOSITE FILTERING QUERIES

Q1: (Symbol = TDDD) | (Symbol = ABTE)
Q2: (Industry = IND00) | (Industry = IND01) | (Industry = IND02)
Q3: (Industry = IND00 & Year = 1999) | (Industry = IND01 & Year = 2000) | (Industry = IND02 & Year = 2001) | (Industry = IND03 & Year = 2002)
Q4: (Symbol = ACSEF & CurrentPrice < 116.0) | (Symbol = TDDD & CurrentPrice > 177.0) | (Volume < 205697)

same queue. We do not have to provide synchronized access to the single queue for the concurrent subscribers, ensure that all readers get fair access to the queue, or maintain the message in the queue until everyone has read it.

Individual queues in the Queue Management Subsystem have the overwrite property that takes effect when the queue becomes full. In such a case, the queue tail starts pointing at the head and the most recent messages are still stored in the queue. In other words, when the queue gets full, its elements are overwritten starting from the tail and progressing towards the head of the queue. The most common solution of emptying the queue when it becomes full is not acceptable in a pub/sub environment as messages may be lost. If a subscriber has slow link, we can increase the queue size, if possible to alleviate the problem. However, if a subscriber has slow link, only that particular subscriber will experience the gradual eviction of older messages from its queue due to congestion and the other (possibly faster) subscribers will not be affected by virtue of the individual dedicated queue model.

V. DEPLOYMENT AND EXPERIMENTAL RESULTS

We have deployed the subscriber-centric content filtering service on the Linux cluster of workstations in our Computer Science department. For the purpose of conducting accurate experimental measurements of distributed service deployment we have developed the automated scripting solution that records the runtime information of the individual broker nodes. We performed our experiments using the local environment as the hardware and where the load ratio is controllable, instead of the cloud-based environment like Amazon Web Services (AWS) where machines are offered in the Virtual Machine (VM) mode that limits the performance scalability. As described in Section II we have used the scenario depicted in Figure 1 where the main stream is received from the external web service via the HTTP agent. We have used the Yahoo! Finance service to receive the continuous XML stream of quotes for the New York Stock Exchange (NYSE) and S&P 500 companies [14] in near real-time. Thus, the sample XML dataset can be obtained through querying the Yahoo! web service [15] resources. TeleScope [5] distribution also provides similar NASDAQ stocks dataset as well as TPC-H Relational Database Benchmark dataset obtained from the University of Washington XML Data Repository [16]. The supplied datasets contain the XML messages that are on an average 5 to 6 times shorter than the messages processed in our tests that are obtained from the actual Yahoo! Finance service in near real-time.

The stream is received via the HTTP REST API facility using wget agent, saved on disk and then processed by the XML transformation agent to transform messages into TeleScope compliant XML message format. The message transformation essentially just adds the length XML attribute to the opening XML tag of the message to indicate the start and end of the message in the XML stream. The transformed messages are then being read by the root TeleScope instance and further distributed to the subscribers residing on separate machines in the Linux cluster. During our experiments we focused on measuring and assessing the following key indicators of the service: (i) filtering capabilities, (ii) raw message forwarding performance (messages forwarded per minute) during variable concurrent subscriptions , and (iii) evaluation of fair queuing facility for effective replication mechanism. We cover each of the indicators in the following subsections.

A. Filtering Capabilities

We have deployed the tree based content distribution/dissemination topology where the root broker receives live stream from the disk (received from Yahoo! Finance service using YQL [15] REST query every 5 minute intervals and prepared for consumption by the transformation agent). We have deployed up to 128 subscribing brokers residing on different machines in the cluster that subscribe to the root broker at the same time and receive the unfiltered NYSE quotes for S&P 500 companies. Each broker then registers a filtering query at its end to filter and save only a specific company ticker symbol quote with supplemental filtering query conditions. The typical filtering query executed at each subscribing broker instance is depicted in Table V.

TABLE V. SAMPLE S&P 500 QUOTE FILTERING QUERY BASED ON TICKER SYMBOL

Q1: (Symbol = MSFT & Currency = USD & StockExchange = NYSE & Ask > 0) | (Symbol = MSFT & Currency = USD & StockExchange = NasdaqNM & Ask > 0)

Several elements are chained in the OR relation to form a composite filtering query that is often required to filter the necessary messages from the general XML stream. All the messages matching the filtering queries have been saved on a disk and examined on the exact match - the filtering engine captured only the stock quote messages corresponding to the filtering conditions without any false positives.

B. Raw Message Forwarding Performance

We have tested the forwarding rate of serving variable number of concurrent subscribing instances at a single publishing broker delivering constant flow of messages to its subscribers. The specification of nodes involved in the benchmarking is depicted in Table VI.

TABLE VI. NODE SPECIFICATIONS

Hardware	Description
CPU	Intel(R) Xeon (R) E545 @ 3.00 GHz
CPU Cores	8
Filesystem	RAM based tmpfs
RAM	16 GB
OS	Fedora 20 with Linux kernel 3.16.3-200
Inter-node network latency	0.1 ms
Avg. XML message size	3.5 KB
Num. XML messages	500 000
XML Dataset size	1.5 GB

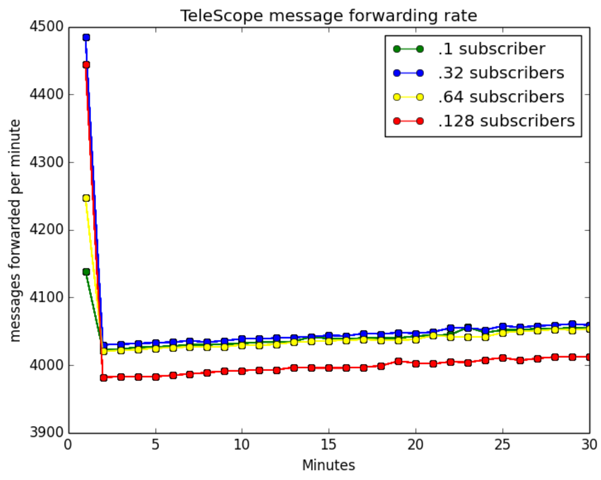

Fig. 5. Message Forwarding Performance with Subscribers Located on a Single Separate Node

The performance measurements have been conducted to eliminate the potential disk/network I/O bottleneck. Therefore the root publishing broker has been serving the XML messages directly from the local data file (residing in the relatively fast RAM based tmpfs) publishing 500 000 XML NASDAQ quotes (file size in tmpfs: 1.5 GB) to the subscribers. Subscribers have also been saving received messages into their respective tmpfs. The performance evaluation under various subscriber loads is depicted in Figure 5. In this experiment all subscribers have been located on a single separate node.

Clearly TeleScope is performing equally well with a single subscriber as well as with 128 concurrent subscriptions originating from a single node. At the same time the total CPU utilization rate at the publishing broker was reaching 100% (occupying 7 out of 8 CPU cores on the machine) with 128 concurrent subscribers on the machine with hardware specifications outlined in Table VI. The memory utilization of a running instance per distinct mode of operation (publisher/subscriber) using the static buffer approach (instead of relying on dynamic memory allocation/deallocation that yields less memory consumption but is not reliable in the long run on overloaded instances) is depicted in Table VII.

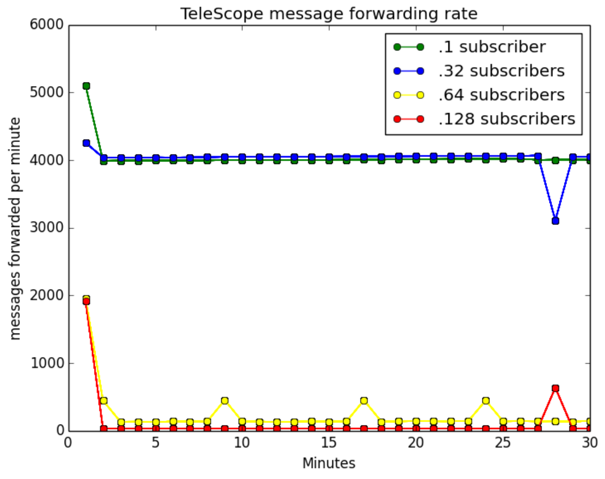

Fig. 6. Message Forwarding Performance with Subscribers Located on Distinct Separate Nodes

The results for the case where subscribers are residing on separate machines are shown in Figure 6. This scenario is close to realistic deployment cases and clearly shows the linear scalability of forwarding rates. Note the drastic difference in message forwarding rates at the threshold of 64 subscribers (results confirmed through numerous repetitive experiments) raise questions at the OS network stack level of XML stream delivery. This may also explain the difference in forwarding rates between running the same number of subscriptions on the same node and in a distributed fashion as depicted in Figure 6.

Real-world deployments often do not exhibit the constant influx of messages at massive rates and therefore do not necessarily require the high forwarding rate to be constantly maintained at publishing broker instance. In fact in our deployment we have been receiving the XML stream every 5 minutes (from Yahoo! Finance) simply because that is an average update rate at the source provider itself. In this scenario the amount of messages being pushed to the subscribers is usually well under 100 MB (often just a couple of MB or even smaller) and not gigabytes as used in our performance evaluation. Therefore query filtering capabilities of the underlying engine often play a more important role than the high forwarding performance in real-world deployments.

C. Replication Effectiveness

The effectiveness of replicating XML messages has been tested under various subscriber loads and is depicted in Figure 7. The experiments have been conducted using tree-like hierarchy with the leaf subscriber broker nodes connected to the single upstream root publishing broker instance that disseminates the stream to the downstream nodes. In our deployment scenario all subscribers have been located in the same network with the same Round Trip Time (RTT) distance from the root

TABLE VII. MEMORY UTILIZATION WITH STATIC BUFFER APPROACH

Mode of operation	RAM utilization	Num. of subscriber queues	Num. of messages per queue
Publisher	800 MB	128	1000
Subscriber only	68 MB	0	0

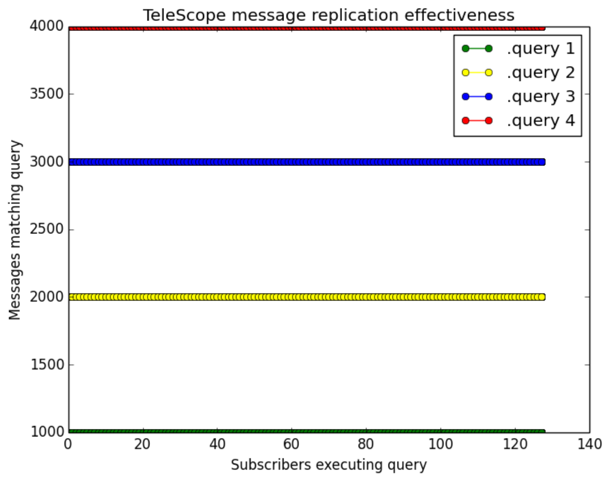

Fig. 7. Replication Efficiency

broker so all of them will receive identical messages. The subscribers store the message on disk if message matches the filter condition for every type of filtering query registered at the endpoints. Therefore selective dissemination/replication of XML stream at the publishing broker was performed using fair queuing facility of TeleScope in an effective manner. The following sample filtering queries have been registered on all the child brokers in succession:

- Q1: Symbol = MSFT
- Q2: Symbol = MSFT | Symbol = ABT
- Q3: Symbol = MSFT | Symbol = ABT | Symbol = BLL
- Q4: Symbol = MSFT | Symbol = ABT | Symbol = BLL | Symbol = EBAY

There are 500 company quotes in the dataset of 500,000 XML messages as outlined in Table VI. Therefore each company quote occurs in 1000 messages. Every subsequent filtering query adds additional company quote to the expression.

VI. DISCUSSION OF ARCHITECTURAL PROPERTIES

Based on our experiments, we briefly outline a number of aspects of the deployed architecture and point to potential extensions/improvements of our current solution. We have observed the following.

Scalability with a large number of subscribers: The proposed filtering and dissemination service has a possible scalability limitation at the level of the root publishing broker. We have been able to serve 128 subscribers with a consumption of approximately 7 CPU cores out of 8 present. However the results may vary in a scenario when subscribers have longer distance from the root broker in terms of number of network hops and RTT delays. The placement of subscribers further away in the network from the root node generally decreases the CPU utilization and increases the throughput and the number of connections served due to the fact that more network I/O has to be performed and less CPU operations are necessary – it takes time to send a message from the queue and get it delivered to the subscriber across the network before the next message is dispatched. The provisioning of root instance on a more powerful hardware ultimately increases the scalability. Clearly the availability of several CPUs with tens of CPU cores per CPU as well as hundreds of gigabytes of RAM storage for in-memory queues will allow the root node to scale with increased number of subscriber connections.

TeleScope currently employs the thread pool approach to serving subscribers – a separate thread is dispatched per connection with the dedicated in-memory queue per thread; this obviates a need for mutex locks and improves performance. In short, each CPU core is occupied by a single reader thread. In the event of excessive number of subscribers that outgrows the number of available CPU cores the instance relies on OS level scheduling for dissemination.

In traditional web server or web caching architectures, clients may be idle and may not send continuous content requests to the server which regulates the amount of content sent back to the client. Our approach, in contrast, assumes constant delivery of messages to the subscriber. This particular communication pattern justifies the allocation of a separate reader thread per subscriber since every reader dispatches XML messages to subscriber in constant time unless the subscriber is disconnected from the broker. However for long-running, partially inactive connections (e.g. long-polling notification requests), the HTTP transaction-oriented communication high performance event notification interfaces such as epoll or kqueue are generally preferred over the vanilla multi-threaded approaches [17]. In short, the thread pool model may be less efficient in some scenarios. However, it works well for our service architecture that allows vertical scaling of the dissemination with large amounts of XML messages in the stream.

Self-configurable topology: Our current architecture does not support the control channel communication between the broker nodes for the purpose of dynamic reconfiguration or failover resiliency. This is a complex problem [8], [4] that currently does not fall in the scope of our subscriber-centric filtering architecture. However a possible attempt to address the aspects of service discovery could be done by introducing the information about the current active filtering query on the instance. That is an incremental effort since each TeleScope instance operating in server mode already reports its run-time information via a status thread. The addition of query information will enable the creation of rudimentary service discovery facility in large filtering overlays.

VII. RELATED WORK

Data stream management systems such as [18] [19] [20] [21] were proposed as extensions to the relational database systems in an effort to support stream processing. They provide very expressive query languages, but are not scalable and cannot support a large number of concurrent queries. Scalability and load tolerance have been addressed in systems such as Borealis [19], TelegraphCQ [20] and Cayuga [21] and focuses on combining pub/sub concepts with relational data stream paradigm. Such systems have their own stream representation formats which restricts their interoperability.

Classic pub/sub systems provide customized notifications to users in a distributed environment by virtue of having loose coupling between information producers (publishers) and consumers (subscribers) [1]. Such systems trade expressiveness of query languages for reasons of performance. When well engineered, they exhibit very high scalability in both the number of queries and the stream rate. Twitter, geomarketing, traffic and weather alerts, emergency response services based on mobile social networks use pub/sub as the underlying technology. These systems are large (e.g. Twitter is estimated to handle over 400 million tweets daily), geographically distributed and largely subscription based. Subscriptions in such systems is often simple and short-lived.

Most pub/sub systems are either *content-based* where subscribers receive notifications when the content of the message matches their interest or *topic-based* when the topic of the publication matches their interest. One of the main incentives for content-based pub/sub is to enable delivery of relevant and meaningful notifications through rich and expressive subscription languages. Sophisticated subscription management strategies are usually required to determine matching subscribers and subsequently perform routing only of the relevant events; runtime overheads for subscription insertion/removal and event matching in content-based model are therefore much higher than topic-based implementations [1][22]. Our work also is on content-based pub/sub where the data streams are in XML format.

Content-based XML stream processing research in [7], [10], [8] has closest resemblance to our work. XML content routing in [8] constructs an overlay network that transports XML streams. It is comprised of XML routers that have XML engines based on XQuery that evaluate each received XML packet against all output link queries. TeleScope is able to form the overlay network for XML stream dissemination/replication similar to the work [8], but provides *subscriber-centric* content filtering. We do not focus on forming intelligent meshes with initialization, maintenance, repair and dynamic reconfiguration capabilities in the face of the incremental overlay node failures. We focus on solving the task of content filtering for reasons of efficient dissemination.

ONYX [7] is a system also based on an overlay network comprised of YFilter [10] XML query engine broker nodes. It attempts to address the problem of efficient XML content dissemination on the Internet scale [7]. Our work on XML stream dissemination does not use XPath/XQuery, but simple filtering constructs needed for efficient dissemination. Our content filtering service does not attempt to incorporate the complex message transformations, concurrent multi-query registration, and sharing at a single broker node. Instead, we distribute the work of content filtering, dissemination, and message transformation across various service agents for reasons of efficiency. ONYX's [7] approach is based on performing incremental message transformations to reduce message size and does not transmit the entire XML message published by the data source. ONYX delivers only the parts of the message actually selected by the data sinks' subscribing queries. Our work targets applications in which the subscribing data sinks need the entire contents of the messages [4]. We do not compare our performance to the YFilter as we support different types of operators and expressions on the messages [23], and also deal with complete messages.

Our work on XML content filtering is fundamentally different from XML query processing or pattern matching. The main goal of XML query processing is to find specific parts within XML message/document which match a query by building suitable indexes over it. The majority of XML filtering approaches use the twig patterns using XPath expressions to represent entities such as user profiles [23]. Published documents are matched against XPath queries and delivered to the interested subscribers [4].

Our current work focuses on real-time filtering of every XML message in the stream and uses an alternative approach without reliance on XPath framework. In contrast to XPath, the developed semantics allows the formulation of boolean logic expressions including support for negations. The work by Fegaras et al. [24] also relies on XQuery framework and its query processing framework targets the querying of fragmented XML data from multiple documents in the same stream, message transformation such as repetition and replacement of fragments, introduction of new fragments or deletion of invalid ones [24]. Our work targets different requirements. We target the processing of relatively small (1 KB to 256 KB) XML messages (typical for most of Internet applications) which facilitates efficient filtering performance with a large number of connected subscribers and high XML stream arrival rates, as in [7] and [8]. Moreover, in the work by Fegaras et al. [24], the publisher disseminates to concurrent subscribers using multicast, whereas we have separate queues for the subscribers. Finally, Fegaras et al. do not focus on real-time in memory processing of XML streams.

Value-based predicate filtering of XML documents in [23] and [13] using twig patterns has limited expressiveness of XPath expressions which is needed for efficient XML content filtering. The authors propose the extension of XPath to build the comprehensive XML filtering system that evaluates value-based predicates in twig patterns and matches their structure holistically. Our solution does not rely on XPath and its twig patterns. Moreover, we provide a comprehensive pub/sub infrastructure instead of just a filtering engine. Both [23] and [13] use the similar XML message access approach – incoming XML documents that need to be filtered are initially parsed using a SAX parser. We, on the other hand, use a DOM method that does not incur significant memory overhead for short XML messages used in our deployment scenarios.

Our work on providing a language bears resemblance to work in PADRES [22] content-based publish-subscribe system. The filtering query expressions in both solutions are formulated in a very similar manner, but the filtering capabilities in PADRES are not focused on XML streams [4].

Stream processing research efforts such as StreamHub [9], Timestream [25], S4 [26] and System S [27] are oriented towards massively scalable processing of data streams in the cluster environments where individual processing operators are usually distributed across the nodes in the directed acyclic graph. We do not address the problem of distributing the stream computation across servers for load sharing, but focus on efficient stream dissemination to the subscribers based on content-filtering.

VIII. Conclusion and Future Work

Many applications generate streaming data at various sources that must be efficiently distributed and processed in near real-time by consuming applications. Towards this end, we propose a configurable content filtering broker architecture for efficient dissemination of XML streaming data to various subscribers. Our approach supports interoperability and allows streams generated from various sources to be content filtered so that applications are given only the relevant data. We developed a prototype and performed experiments to demonstrate the feasibility of our approach. Our performance on the prototype demonstrates that such an approach will scale to real-world applications. In future, we plan to enhance the filtering engine in TeleScope so that we can process parameterized aggregates and window operators in the filtering query, introduce full string search capability, and do unbounded processing of XML attributes. We also plan to add the support for rule model so that it can be used for complex event processing. We also plan to address issues pertaining to security, such as specifying and enforcing content-based and context-based access control on semi-structured data streams. Specifically we anticipate the enhancement of the current filtering engine with functionality to process XML policy messages for the purpose of distributed policy orchestration [28].

Acknowledgement

The work was supported in part by a funding from NIST under Award No. 70NANB15H264, from NSF under Award No. CCF-1018711, and from an internal grant from the Colorado State University,

References

[1] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, 2003.

[2] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "PNUTS: Yahoo!'s Hosted Data Serving Platform," *Proc. of VLDB*, vol. 1, no. 2, pp. 1277–1288, 2008.

[3] P. A. Bernstein, N. Dani, B. Khessib, R. Manne, and D. Shutt, "Data Management Issues in Supporting Large-Scale Web Services," *IEEE Data Engineering Bulletin*, vol. 29, no. 4, pp. 3–9, 2006.

[4] G. Li, S. Hou, and H.-A. Jacobsen, "Routing of XML and XPath Queries in Data Dissemination Networks," in *Proc. of ICDCS*, 2008, pp. 627–638.

[5] K. Belyaev, "TeleScope - XML Data Stream Broker/Replicator," http://sourceforge.net/projects/telescopecq, accessed: 2-September-2015.

[6] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, and J. Siméon, "XML Path Language (XPath)," *World Wide Web Consortium (W3C)*, 2003.

[7] Y. Diao, S. Rizvi, and M. J. Franklin, "Towards an Internet-Scale XML Dissemination Service," in *Proc. of the VLDB*, 2004, pp. 612–623.

[8] A. C. Snoeren, K. Conley, and D. K. Gifford, "Mesh-Based Content Routing using XML," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 160–173, 2001.

[9] R. Barazzutti, P. Felber, C. Fetzer, E. Onica, J.-F. Pineau, M. Pasin, E. Rivière, and S. Weigert, "StreamHub: A Massively Parallel Architecture for High-Performance Content-Based Publish/Subscribe," in *Proc. of DEBS*, 2013, pp. 63–74.

[10] Y. Diao and M. Franklin, "Query Processing for High-Volume XML Message Brokering," in *Proc. of the VLDB*, 2003, pp. 261–272.

[11] D. Veillard, "The XML C Parser and Toolkit of Gnome," http://www.xmlsoft.org/, accessed: 2-September-2015.

[12] W3C, "The Document Object Model," http://www.w3.org/DOM/#what, accessed: 2-September-2015.

[13] J. Kwon, P. Rao, B. Moon, and S. Lee, "FiST: Scalable XML Document Filtering by Sequencing Twig Patterns," in *Proc. of the VLDB*, 2005, pp. 217–228.

[14] Wikipedia, "List of S&P 500 Companies," http://en.wikipedia.org/List_of_S\%26P_500_companies, 2015, accessed: 3-February-2015.

[15] Yahoo!, "Yahoo! Query Language," https://developer.yahoo.com/yql, accessed: 2-September-2015.

[16] U. of Washington, "XML Data Repository," http://www.cs.washington.edu/research/xmldatasets/, accessed: 2-September-2015.

[17] M. Welsh, D. Culler, and E. Brewer, "SEDA: An Architecture for Well-Conditioned, Scalable Internet Services," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 230–243, 2001.

[18] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom, "STREAM: The Stanford Data Stream Management System," in *Processing High-Speed Data Streams*. Springer-Verlag, 2004.

[19] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina *et al.*, "The Design of the Borealis Stream Processing Engine." in *Proc. of CIDR*, 2005, pp. 277–289.

[20] S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah, "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World," in *Proc. of CIDR*, 2003.

[21] A. J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, W. M. White *et al.*, "Cayuga: A General Purpose Event Monitoring System," in *Proc. of CIDR*, 2007, pp. 412–422.

[22] H.-A. Jacobsen, A. K. Y. Cheung, G. Li, B. Maniymaran, V. Muthusamy, and R. S. Kazemzadeh, "The PADRES Publish/Subscribe System." in *Principles and Applications of Distributed Event-Based Systems*. IGI Global, 2010.

[23] J. Kwon, P. Rao, B. Moon, and S. Lee, "Value-Based Predicate Filtering of XML Documents," *Data & Knowledge Engineering*, vol. 67, no. 1, pp. 51–73, 2008.

[24] L. Fegaras, D. Levine, S. Bose, and V. Chaluvadi, "Query Processing of Streamed XML Data," in *Proc. of CIKM*, 2002, pp. 126–133.

[25] Z. Qian, Y. He, C. Su, Z. Wu, H. Zhu, T. Zhang, L. Zhou, Y. Yu, and Z. Zhang, "Timestream: Reliable Stream Computation in the Cloud," in *Proc. of EuroSys*, 2013, pp. 1–14.

[26] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed Stream Computing Platform," in *Proc. of ICDM Workshops*, 2010, pp. 170–177.

[27] H. Andrade, B. Gedik, K.-L. Wu, and P. Yu, "Processing High Data Rate Streams in System S," *Journal of Parallel and Distributed Computing*, vol. 71, no. 2, pp. 145–156, 2011.

[28] J. Singh, J. Bacon, and D. Eyers, "Policy Enforcement within Emerging Distributed, Event-Based Systems," in *Proc. of DEBS*, 2014, pp. 246–255.