Bistro Data Feed Management System

Theodore Johnson Vladislav Shkapenyuk Divesh Srivastava
AT&T Labs - Research
180 Park Avenue. Bldg. 103
Florham Park, NJ, USA
{johnsont, vshkap, divesh}@research.att.com

ABSTRACT

Data feed management is a critical component of many data intensive applications that depend on reliable data delivery to support real-time data collection, correlation and analysis. Data is typically collected from a wide variety of sources and organizations, using a range of mechanisms - some data are streamed in real time, while other data are obtained at regular intervals or collected in an ad hoc fashion. Individual applications are forced to make separate arrangements with feed providers, learn the structure of incoming files, monitor data quality, and trigger any processing necessary. The Bistro data feed manager, designed and implemented at AT&T Labs-Research, simplifies and automates this complex task of data feed management: efficiently handling incoming raw files, identifying data feeds and distributing them to remote subscribers.

Bistro supports a flexible specification language to define logical data feeds using the naming structure of physical data files, and to identify feed subscribers. Based on the specification, Bistro matches data files to feeds, performs file normalization and compression, efficiently delivers files, and notifies subscribers using a trigger mechanism. We describe our feed analyzer that discovers the naming structure of incoming data files to detect new feeds, dropped feeds, feed changes, or lost data in an existing feed. Bistro is currently deployed within AT&T Labs and is responsible for the real-time delivery of over 100 different raw feeds, distributing data to several large-scale stream warehouses.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Distributed systems.

General Terms

Algorithms, Management, Performance, Design.

Keywords

Data feed management, distributed systems.

1. INTRODUCTION

Large organizations with diverse operations, such as AT&T but including many others, generate a wide variety of continuous streams of data files from operations at geographically distributed locations. These data files must be brought together and correlated for an organization to make sense of and to control its operations. The stream of data files resulting from a particular type of activity is called a *data feed*, and the process of collecting, managing, and distributing a collection of data feeds is called *data feed management*.

Let's consider a simple example. A shipping company might collect information about how packages are managed within its operations. There are a number of data sources within the company infrastructure that generate a variety of data feeds relevant to package management. One data feed originates from shipping centers and contains a stream of log files with package drop-offs records. Another data feed contains the logs of bar code scans of packages as they move between trucks and warehouses. A third data feed contains GPS reading from delivery trucks. The final data feed contains the electronic delivery signatures collected from customers.

To be useful, these data feeds need to be gathered and delivered to company analysts. The gathering process can be non-trivial, because the data sources are geographically dispersed and some are only sporadically connected to the network (e.g., handheld barcode scanners). Once collected, the data needs to be distributed to company analysts. One analyst group might be based in Atlanta and specialize in marketing. These analysts would be primarily interested in the package drop-off data feed. Another group might be based in Dallas, and primarily interested in optimizing operations. This group would subscribe to the barcode scan and truck GPS data feeds. A third group might run a corporate warehouse and subscribe to all data feeds.

The ability to manage real-time data feeds can provide a competitive advantage to the company. For example, if the delivery data feed can be made real-time, the company can provide real-time delivery alerts to its customers.

Another example that we are going to use throughout this paper is inspired by the AT&T network measurement infrastructure. Running large networks and maintaining service infrastructures requires collecting massive amounts of data from all the network elements [13], including network element configuration and topology, network events, network element fault logs, workflow logs, performance data, and so on. SNMP pollers are important components of the overall network measurement infrastructure.

They are responsible for periodic querying of various network elements (e.g. routers, VoIP equipment) and collecting a variety of statistics describing the state of each measured element. Some example statistics include router CPU load, memory utilization, number of bytes and packets send and received per second, link utilization and link loss data. For every measurement interval an individual poller generates a set of files each containing one kind of statistics. Since new sets of files are generated continuously by a poller, they form an aggregate data feed that we will call a *feed group*. Aggregate feed groups can be further combined into even larger feed groups, for example when measurement data from several different pollers is combined.

In order to take advantage of the wealth of information generated by various pollers, collected data feeds need to be delivered to several groups within a company responsible for monitoring and analysis of network data. Different groups within a company might be interested in different subsets of the data collected by SNMP pollers. For example a group responsible for customer billing applications is interested in bytes per second (BPS) data combining BPS measurement from all the available pollers. Another group responsible for network capacity planning is interested in all available statistics to be able to make future projections. Generally, any application can define a set of predicates that will select a subset of the full set of available data feeds that this particular application needs. We call these application-defined data feeds *consumer feeds* to differentiate them from *source data feeds* generated by data sources such as pollers.

Similar to the shipping company application, the ability to deliver data feeds in real-time is critical for many classes of applications. For example, a network fault detection application needs a logical feed containing link fault data, with real-time propagation to minimize the customer impact. Network visualization applications are also extremely sensitive to any delays introduced by the feed delivery infrastructure.

In the absence of available software systems designed specifically for data feed management, many organizations are forced to develop homegrown feed management systems. Designing a robust system capable of handling real-time delivery of massive data feeds to large number of applications requires solving a number of technical challenges ranging from designing communication protocols and transfer scheduling algorithms to developing feed monitoring and analysis tools. The main goal of the *Bistro* project at AT&T Labs was to design and build a software platform for managing all the aspects of feed management. While the primary focus of the *Bistro* data feeds management system was to handle industrial-scale data feed management problems we encountered while building data systems for AT&T, most of the issues addressed by the system are generally applicable to any large organization. Bistro servers currently manage over 100 data feeds, delivering up to 300 gigabytes of data per day to a number of customers in real-time, while also providing real-time delivery alerts.

In Figure 1 we show a high-level picture of how a Data Feed Management System (DFMS) fits into data ecosystem within a large organization. Distributed data sources generate a number of source data feeds that are delivered over the network to DFMS. DFMS performs a matching between incoming data files and a set of consumer data feeds defined in the system. Subscribers (e.g applications consuming the data) subscribe to one or more consumer data feeds managed by Data Feed Management System. The DFMS is then responsible for reliable real-time delivery of consumer feeds to all the interested subscribers.

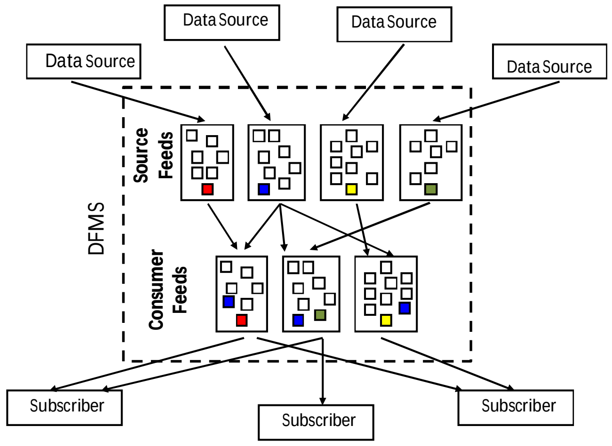

Figure 1. Data Feed Management System

The rest of the paper is organized as follows. We give detailed discussion of the major challenges in the area of data feed management in Section 2. In Section 3, we outline the general architecture of a Bistro server and describe all the major components of the system. In Section 4 we describe Bistro feed delivery mechanism, including communication protocols, scheduler and scheme for ensuring reliability. Section 5 describes the feed analysis system and monitoring tools. We discuss relevant related work in Section 6. We present our conclusions in Section 7.

2. CHALLENGES IN DATA FEED MANAGEMENT

In the past the design of Data Feed Management Systems received very little interest from academia and industry, forcing large organizations to implement and deploy a multitude of homegrown DFMSs. Most of such systems face the same set of technical challenges that need to be resolved in order to robustly handle massive data feeds, with large number of feeds and subscribers while providing timeliness guarantees. In this section we will enumerate the main challenges facing the designers and implementers of Data Feed Management Systems. We will cover the issues in feed metadata discovery, dealing with feed evolution over time, building a feedback mechanism between DFMS and subscribers, ensuring reliable feed delivery, real-time transfer scheduling and support for notifications and triggers.

2.1 Feed Metadata Discovery

In order to be able to use a data feed, an application needs to perform a significant amount of groundwork to discover all the metadata necessary for the correct interpretation of data files.

Feed file naming convention. Typically a data source generating a source data feed follows a certain naming convention for the output files. For example an SNMP poller generating a history of alarms for a set of monitored routers will generate output files using *ALARMHISTORYpoller_idTS.gz* naming convention, whrere *poller_id* is an integer and *TS* is a timestamp including year, month, day, hour and minute.

In general, a variety of information can be encoded in a file name including:

- One or several timestamps describing measurement interval
- Names of the objects generating reports
- Attributes of the object generating reports (e.g location, software version, etc)
- File format (e.g csv, tar, gz)
- Hierarchical organization of the feed files. Describes hierarchical directory structure which is used by the source to organize files (e.g *YYYY/MM/DD/filename* or *YY-MM-DD/filename*)

File arrival patterns. Real-time applications relying on data feeds need to make certain assumptions about tardiness of the data arriving as a part of data feed. Some of the feed attributes that needs to be discovered in advance include:

- How frequently new feed data are being generated by the source
- Whether feed data are being generated for fixed-size or arbitrary sized time intervals
- What is the maximum delay between a feed data record representing a particular event and the time when feed file with that record is generated.
- Are feed files arriving in order they were generated, or can they potentially arrive out-of-order due to nature of wide-scale file distribution and communication failures.

File format specification. There is a wide variety of file formats used to encode feed data and applications need to be supplied either an exact format specification required to parse the file data records or a data parser itself.

Feed data semantics. An exact semantic interpretation for each field of feed data record will need to be agreed on, including data types, domains, encodings, etc.

Even though ideally all the metadata information would be carefully documented and agreed upon between data sources and prospective subscribing application, in the real world this information is frequently missing and has be discovered by applications and DFMS managing the feed. In the rest of the section we will go over some of the reasons why feed metadata information might not be available and main technical challenges involved in recovering lost metadata.

2.1.1 Incomplete or Missing Feed Metadata

In our experience at AT&T Labs we observed several reasons why feed metadata information might not be available:

Aggregated data feeds. Many real-world data feeds are not generated by a single source, but rather composed from collection of several separate subfeeds providing complimentary information. Consider, for example network measurement infrastructure described in Section 1. A source feed with router memory utilization data could be composed from several subfeeds generated by individual SNMP pollers. Exact number of pollers may change over time with new pollers coming in and old ones going offline or being temporarily unavailable. In addition to combining several uniform feeds, it is common for feed providers to package large number of related feeds together and deliver it to a Data Feed Management System as one large feed group. For example an SNMP measurement feed could contain a large number of individual subfeeds with router CPU load, memory utilization, number of bytes and packets send and received per second, link utilization and link loss data. Some of the source data feeds we are working with within AT&T network measurement infrastructure contain over a hundred loosely related individual subfeeds. Again, the exact composition of large aggregated source data feeds more often than not is not available to DFMS and interested subscribers and tends to evolve over time, which makes extracting individual feeds and discovering associated feed metadata a challenging task.

Lack of direct communication with feed sources. It is very common that the source of the data feed is managed by a separate organization or a different business unit within a company. Different sub-organizations within a large organization have different priorities and often only loosely coordinate with each other. There is very little incentive for an organization with no contractual obligations to invest time in documenting and sharing the metadata about the source feeds it produces.

Furthermore, even the sources themselves may not have full information and would need to go through the documentation of whatever software was used to generate the feed files to discover the metadata associated with the source feed. Whenever there is a change in the software that generates the feed that results in changes in the structure of the feed files, the sources are not always aware of all the subscribers that rely on their data and are not under any obligation to inform them about the changes.

2.1.2 Browsing-based metadata discovery

Given the aggregate nature of data feeds and lack of source support for metadata discovery process, the task of metadata discovery falls on individual subscribers. In the absence of authoritative data feed managers, the only feasible way to discover the metadata is by browsing some history of the data feed and making reasonable guesses. The easiest case for a subscriber is when a feed obtained from a feed provider contains uniform data and is not a collection of disjoint feeds. For example, consider the following stream of daily files:

Poller1_router_a_2010_12_30_00.csv,gz
Poller2_router_a_2010_12_30_00.csv,gz
Poller1_router_a_2010_12_30_01.csv,gz
Poller2_router_a_2010_12_30_01.csv,gz
Poller1_router_a_2010_12_30_02.csv,gz
Poller2_router_a_2010_12_30_02.csv,gz
…
Poller1_router_a_2010_12_30_24.csv,gz
Poller2_router_a_2010_12_30_24.csv,gz

If sufficient history of feed files is available, subscriber can make a reasonable assumption that the feed contains measurement data obtained from *router_a* queries by two pollers (*Poller1* and *Poller2*) generating hourly reports in gzip-compressed csv format and saved in files with names of the form *Poller[1-2]_router_a_YYYY_MM_DD_HH.cvs.gz*. Even in this simple case it is very easy for subscriber to make a number of incorrect assumptions and extract wrong feed metadata based on small data sample:

- assumption that all the measurement data comes from only one router
- assumption about the number and names of the pollers
- assumption that files will always arrive on hourly basis and that *YYYY_DD_MM_HH* part of a filename is correlated with the polling interval
- assumption that all the files produced by the poller will have the same comma-delimited format and will be compressed

In a typical case of a more complex aggregate feed composed of hundreds of individual subfeeds, discovering the metadata for every included subfeed is even more prone to incorrect guesses. These incorrect assumptions about the structure of the individual feeds can be discovered arbitrarily late when users of applications subscribing to affected data feeds start complaining about incorrect or missing data in application-generated reports.

In addition to technical challenges related to extracting correct feed metadata based on limited sample of the data files, there is also a significant performance overhead that browsing directories places on a feed provider. Even through modern filesystems can achieve very high data throughput levels, serving file metadata is always a bottleneck due to a more significant synchronization overhead. The overhead is even more pronounced on distributed filesystems where metadata operations require significant network overhead. It is also not uncommon for feed provider to not offer a filesystem interface instead relying on either database query interface or SOAP-based web interface for data retrieval that does not always naturally support exploratory queries.

2.1.3 *Feed evolution*

A significant challenge during browsing-based feed metadata discovery process is to generate feed definitions that are specific enough to exclude the files that do not belong to a feed but at the same time generic enough to be robust to future changes. Many real-world data feeds are highly dynamic and undergo many changes during a feed lifetime, in filename structure, file arrival patterns as well as in the content of the data files themselves. As a result an old feed definition could either start missing new files that no longer match the pattern, or could start matching files that a subscriber doesn't want. Depending on sophistication of monitoring tools on subscriber side these problems can go undetected for a long time and can lead to dirty and missing data.

2.1.3.1 *False negatives in file classifications*

False negatives in feed file classification generally occur whenever an initial feed pattern was too specific to a data sample used to generate the definition. A false negative can occur when one of the following scenarios occurs:

Feed file naming convention changed. Software changes on feed generation side can result in new naming convention adopted for the output files. For example, an SNMP poller that used to generate measurement reports matching the pattern *poller1_YYYY_MM_DD.csv.gz* had a software update that uses a new format of *YYYY/MM/DD/poller1_soft_version.csv.bz2* to name its output files.

More sources are contributing to a feed. If a subscriber makes an assumption that full list of sources generating the feed data is known in advance and hardcodes this list of all sources as a part of feed definition, this definition is going to be vulnerable to false negatives once new sources are introduced. Again using SNMP measurement infrastructure as an example, it is easy to see how the introduction of new pollers can make the definition used in the previous example invalid.

Initial pattern was based on non-representative data sample. This scenario is common for complex data feeds that are composed of several loosely-related subfeeds. In order to generate a correct feed definition, instances of all of the different subfeeds need to be discovered and analyzed. If one of the subfeeds was either not present or not noticed in the sample, it can lead to future instances of the subfeed not matching the feed definition.

Since we cannot make an assumption that initial feed definition will be done perfectly correctly, nor can we expect this definition to remain correct over the lifetime of the feed, a detection mechanism needs to be in place to detect possible false negative and notify the subscribers. Such mechanism cannot be implemented on subscriber side, since the subscriber will not see all the false negatives and may not be aware that some data is missing. The data source also cannot reliably tell which subset of the data a subscriber is really interested in and whether they are getting all the data they need. Therefore, one of the key challenges of building a Data Feed Management System is building a feedback mechanism which allows DFMS to detect possible false negative and suggest revised feed definition and for subscribers to modify feed definitions in response.

2.1.3.2 *False positives in file classifications*

False positives in feed file classification generally occur if the initial feed pattern was too generic so it covers a lot of unwanted files. Below are few scenarios that can lead to generation of false positives:

Use of wildcards in feed definitions to increase the robustness. Given that complex feeds are composed from several subfeeds it is common to future-proof feed definition by replacing some of the object names with wildcards. For example if the full list of future pollers is not known feed subscriber might replace a poller1 in a previous example (*poller1_YYYY_MM_DD.csv.gz*) with * wildcard to allow arbitrary poller name. This will lead to positive matches for all unrelated files which have filenames ending with *YYYY_MM_DD.csv.gz*.

Use of wildcards to simplify the feed definition. Simple feed patterns are generally preferable to complicated ones due to ease of maintenance. Even though for some fields in the filename a full list of possible values is known in advance, an administrator defining the feed is more likely to use a wildcard than enumerating the list of values. This again can lead to unintended matches in the future.

Generally, detecting false positives is very hard on both subscriber and feed provider side. Ideally, incorrectly classified files will be rejected during ETL or data cleanup process on subscriber side; however it is still possible for bad data to go through and remain undetected. For example, if a data feed composed of bytes per second measurement also starts receiving packets per second data with an identical schema, problem detection might be arbitrarily delayed. Furthermore, tracking which source files contributed to poisoned data is highly challenging. Feed providers can detect and refine potentially erroneous feed definition by employing techniques like clustering and outlier detection; however without feedback from the subscribers there is no way to verify whether the refinements are correct. Designing false positive detection and feed refinement mechanisms along with subscriber feedback system is a major challenge in designing a feed management system.

2.2 Feed delivery

Real-time delivery of large volume data feeds to a large number of subscribers with well-defined tardiness creates a number of technical challenges that are unique to a feed management system. In this section we will show the limitation of traditionally employed push and pull-based delivery mechanisms

2.2.1 Pull-based feed delivery

In general, a pull-based feed delivery mechanism requires a subscriber to make an explicit request to retrieve the data items from the feed provider. A feed provider typically exposes the feed files using filesystem interface, database or web-service interface and allows authorized subscribers to retrieve the files or other object using such interface. In our discussion of pull-based systems we are going to focus on systems exposing filesystem interface, however all the drawbacks discussed below apply to database and web-servers as well.

The basic problem with any pull-based system lies in the fact that in order to retrieve a file from a server subscriber needs to be aware of the file's existence. First, a subscriber would need to resolve all the technical challenges we discussed in Section 2.1 associated with discovering the patterns used for feed file generation and estimating frequency of file arrival. Second, a subscriber will need to regularly poll remote directories in which new files are expected to show up in order to detect new files that need to be retrieved. Normally file server directories are designed to provide fast inode lookup rather than fast scan performance. We observed many cases of severe performance degradation as a result of excessive directory scanning. If real-time propagation of feed files is not required, the frequency of performing remote directory listing can be limited to the expected frequency of file arrival. However, for real-time applications that require data propagation with well defined tardiness, the polling needs to be done continuously which places even greater load on remote filesystem.

A well-behaved subscriber trying to limit the load it places on feed provider will try to limit the directory listing operation to a set of directories that contain only the most recent data. However, in real-world applications feed files can arrive arbitrarily late and frequently out-of-order [14] which forces the subscriber to scan even the directories that contain very old data. As a stored feed history stored on a feed provider grows, the cost of the filesystem metadata operations (such as performing directory listing) grows linearly with the history size. Performance considerations force subscribers of pull-based system to impose a limit of out-of-orderness and start ignoring the data that fell out of the recent time window.

Performance penalties of pull-based feed delivery systems become more pronounced in a typical scenario when feeds are shared by a large number of subscribers. Since there is no subscriber cooperation, all of them scan feed directories at the same time, eventually saturating the metadata serving capacity of the filesystem. The performance of file retrieval operation themselves is also suboptimal because file retrieval is done on demand which does not allow a feed management system schedule file retrieval request for maximum temporal and spatial locality of accesses.

2.2.2 Push-based feed delivery

The goal of push-based feed delivery mechanisms is to remove the need for subscribers to perform periodic directory polling by instead delivering the files directly to subscribers that registered interest to a particular feed. Even though this approach alleviates major problems of pull-based systems, a successful implementation of push-based system requires solving a number of technical challenges:

Reliable feed delivery. Data Feed Management System needs to be robust to expected subscriber failures and ensure the guarantee that eventually all the feed files will be transmitted to interested subscribers. Generally, in order to implement a reliable feed delivery mechanism DFMS will need to manage a persistent state tracking file delivery receipts.

Real-time delivery scheduling. A Data Feed Management System needs to guarantee that files will be delivered with a well-defined tardiness. Given limited network bandwidth and local resources available on feed provider side, appropriate transmission scheduling policy will need to ensure such guarantees. It will also need to ensure that no slow or constantly failing subscriber with an ever-growing delivery queue causes the starvation of well-behaving subscribers.

A commonly used implementation of push-based delivery mechanism is based on open-source *rsync* utility [17] which synchronizes the content of two directories while minimizing the amount of data that needs to be transmitted by using delta compression. Since *rsync* itself has no scheduling support, scheduling is implemented using the Unix *cron* facility to periodically schedule rsync jobs for all the subscribers. However, this implementation of feed delivery systems using combination of cron and rsync suffers from serious drawbacks:

1. Lack of subscriber notification forces them to perform periodic directory scans to detect new file arrivals. In addition to the previously outlined performance penalties, the lack of notification also makes it difficult to implement real-time applications that need to invoke some processing immediately after new data becomes available.
2. Rsync stores no state about which files were already delivered to which subscriber, instead relying on both local and remote directory scan to come up with a list of files that need to be transmitted. As stored history grows larger on both source and destination side, the cost of the directory scan grows linearly and completely dominates the actual data transmission time.
3. Loss of control over the directory structure at the destination: Since rsync tries to make the destination directory the same as the source, the destination will be filled with as much data as the source, even if a smaller data window is desired. Rsync makes it difficult for the subscriber to implement a “landing zone”, and combined

with the lack of destination notification makes partial file loading a constant danger.

4. Cron is not very well suited for feed delivery scheduling since it can step on previously unfinished tasks, introduces unnecessary delays (as opposed to triggered processing) and does not provide prioritized resource management.

In Section 4 of the paper we will show how Bistro feed management system implements reliable feed delivery, persistent state management and transmission scheduling that address the issues associated with rsync/cron.

2.3 Notifications/ Triggers

Real-time applications such as visualization systems and streaming data warehouses require immediate notification whenever new feed data is available. As we mentioned previously, any approach relying on periodic directory scans is either too expensive or introduces the unnecessary propagation delays unacceptable for real-time application. A usable Data Feed Management System must implement a scalable but lightweight communication protocol with the subscribers to either notify them that a new file has been delivered or is available to be retrieved from the DFMS. This will allow the subscriber to trigger required data ingest procedures with no further delay.

Even though many applications are interested in immediate notification for each available or delivered file, some applications prefer to be notified on per-batch level. One important class of such applications are streaming data warehouses that maintain a large collection of temporary partitioned materialized views derived from raw feed data. Instead of relying on incremental view maintenance, these warehousing systems use a simpler method of recalculating small set of affected recent partitions. Since a number of files can contribute to an individual partition, it is preferable to invoke the triggered updates only when the raw files contributing to that partition has been received. If a feed management system uses per-file notifications and triggers warehouse update for every received file, it will cause a lot of unnecessary partition recalculations. Similar to a punctuation mechanism in data stream processing systems, the feed manager needs to detect the logical batch boundaries specific to particular application and invoke the notifications accordingly.

Consider a streaming data warehouse that uses 5-min partitions to maintain router memory utilization collected from a number of pollers. If the number of pollers is known in advance, a feed manager can invoke the trigger whenever the same number of 5-minute files was delivered to subscriber. However, this approach is not robust for highly dynamic and frequently unreliable feed when the number of contributing sources can go up or down during the lifetime of the feed. As a result, fixing a size of the batch can introduce delays or unnecessary partition recalculations. Time-based batching that does not exploit the information about the feed composition and application requirements is also prone to delays. A challenge in designing a trigger mechanism for a feed management system is both designing a flexible specification language which allows application to define appropriate notion of a batch, and way for a feed manager to detect the batch boundaries even in the presence of highly dynamic and unreliable feeds.

3. BISTRO SYSTEM DESIGN

The Bistro data feed manager, designed and implemented at AT&T Labs, simplifies and automates complex feed management tasks that we described in previous section providing an effective platform for scalable feed delivery in large organizations. In this section we will outline the general architecture of a Bistro server and briefly describe all the major components of the system. We will also outline structured configuration language used by the system and describe feed analysis and monitoring components.

On a very high level, in a typical mode of operation Bistro data feed manager receives incoming data feeds as a stream of raw files from a set of data sources. Data sources deposit their files in special *landing directories* allocated locally at feed manager site and notify the server that data is ready to be processed. Based on source or subscriber-supplied consumer feed specifications Bistro classifies each incoming file as belonging to one or more registered consumer data feeds, performs filename normalization, optional compression/decompression and places normalized files into a set of *staging directories*.

Using subscriber configuration data which specifies which subscribers are interested in which data feed, the system delivers files from the staging directories to relevant subscribers and invokes subscriber notifications using a trigger mechanism. Records of successful file transmissions are stored in a transactional delivery receipt database to ensure the system's reliability.

Bistro feed analyzer continuously monitors file classification decisions made in a process of file to feed matching to detect and proactively alert the subscribers about possible false positives and false negatives. Additionally, it analyses all the files that did not match any of the registered data feeds to discover new feed definitions. A high-level diagram of Bistro operation is shown in Figure 2.

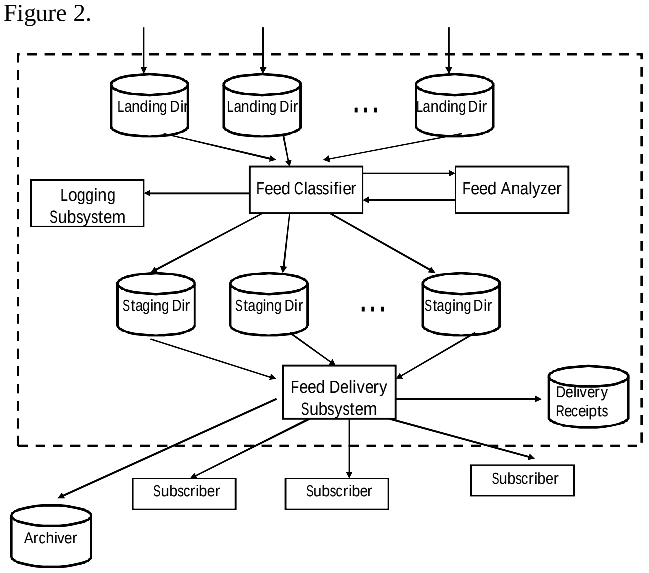

Figure 2. Bistro Server Architecture.

Note that a Bistro server can act as subscriber to another Bistro server allowing the creation of distributed feed delivery network. By organizing Bistro servers into a network of cooperating feed managers we can further increase the scalability of the system and minimize the impact on low-bandwidth network pipes.

3.1 Bistro Configuration Language

As we previously noted in Section 2, many real-world feed management systems are maintained by a collection of ad-hoc scripts encoding which files are supposed to be delivered to which subscriber and using which method. Maintaining such ad-hoc systems is very difficult in the presence of large and highly dynamic sets of data feeds and subscribers with increasingly high probability of configuration mistakes. To address this problem, Bistro uses a well-defined flexible configuration language to formally specify the properties of all managed data feeds and subscribers. We list some of the important configuration parameters below:

Data Feeds. The concept of a data feed is central to Bistro architecture. A data feed is defined as a stream of files produced by one or more data sources (network measurement pollers, databases, data stream management system, other Bistro servers, etc). Data feeds are organized in hierarchical fashion in large feed groups, consisting of individual feeds and other feed groups. An example feed hierarchy for an SNMP feed group containing network router measurement data could look the following way:
SNMP

- ⇨ BPS (bytes per second stats)
- ⇨ PPS (packets per second stats)
- ⇨ CPU (cpu load stats)
- ⇨ MEMORY (memory utilization stats)

Each individual feed (BPS, PPS, …) can be further divided into subfeeds produced by individual pollers (CPU_POLLER1, CPU_POLLER2, etc) forming arbitrarily deep feed hierarchy.

An individual data feed is defined using the following attributes:

1) Feed filename patterns. Bistro uses printf-inspired syntax instead of more traditional regular expressions to define the patterns in names of the files constituting a feed. For example in a pattern *MEMORY%s.%Y%m%d.gz* construct *%s* refers to an arbitrary string and *%Y*, *%m*, *%d* refer to year, month and date respectively. Using this approach we not only simplify feed pattern specifications compared to full-featured regular expressions, but also allow attaching semantic information (such as timestamp) to individual fields within a filename.
2) Filename normalization scheme and compression / decompression options. Often an application prefers to enforce a particular organizational structure to all the files that belong to a data feed, for example organize the files into daily directories or use a directory structure mimicking the feed hierarchy. Of course, organizing files into daily directories would require an understanding of the semantics of timestamp fields embedded into feed file names. The Bistro file normalizer takes knowledge of field semantics embedded in feed patterns to drive the normalization process. Additionally, by using the Bistro configuration language an application is able to expand the data arriving in compressed formats or compress the data before placing it into staging directories.

Subscribers. A subscriber is an application running on remote host interested in receiving files from a set of data feeds. In Bistro configuration language a subscriber is defined using the following attributes:

1) Feeds of interest. Subscriber must specify a set of feeds or feed groups that are in its interest set along with specification where the feed files should be delivered.
2) Feed delivery method. A subscriber can use a variety of methods and protocols to receive data feeds, from pure push based approach to hybrid push-pull based approach where Bistro server notifies the subscriber about availability of new data and leaves it to the subscriber to perform actual data retrieval.
3) Notifications and triggers. Subscribers can specify a lightweight program to be invoked by Bistro server either locally or remotely on subscriber's site whenever new feed data is available. Notifications and trigger invocations can be performed either on a per-file basis or based on a batch of files using time or count-based batch definitions. More detailed discussion of triggers and feed batching support is given in Section 4.1.

3.2 Feed Analysis and Monitoring

The job of Bistro feed classification module is to classify all the files arriving from the data sources into landing directories as belonging to one or more consumer feeds. A classification is performed based on filename patterns defined in formal feed specification using standard regular-expression matching techniques. Even though a classification process based on pattern matching works wells for simple static data feeds with perfectly defined pattern specifications, it is not robust in the presence of many real-world data feeds which are expected to evolve over their lifetimes. Changes in feed compositions, file naming conventions and data formats can all lead to data files being either incorrectly classified as belonging to particular feed or incorrectly excluded from the feed. Furthermore, with the lack of support from data feed providers even initial feed specifications are often incorrect and are likely to generate both false positives and false negatives during feed classification process.

The Bistro *feed analyzer* component is designed to address some of the major issues causing incorrect file-to-feed matching and supports three different modes of use:

Identification of false negatives. Bistro uses structural similarity between unmatched files and already defined data feeds to detect potential false negatives in file classification. Generally, a Data Feed Management System is not allowed to automatically change feed definitions to correct classification errors without confirming the changes with all the affected feed subscribers. Instead, Bistro feed analyzer generates a report specifying files identified as potential false negatives along with suggested changes to feed specification. The ultimate responsibility of approving or rejecting the suggested feed configuration changes is in the hands of feed subscribers.

Identification of false positives. Bistro feed analyzer helps feed's subscribers identify potential false positives in file classification by clustering the files matching particular data feed and identifying outliers that do not share filename structure with the rest of the matching files. Again, rather than modifying feed definitions in-place Bistro generates recommendation for revised feed definitions to be approved or rejected by feed subscribers.

New Feed Discovery. Given highly aggregate nature of many real-world data feeds and difficulty obtaining metadata information describing individual subfeeds, it is not uncommon for a large fraction of incoming data files to be classified as belonging to unknown data feed. Bistro feed analyzer continuously monitors a stream of incoming data files that were not covered by any of the existing feed definitions and periodically generates a list of new feed definitions describing unmatched files. The list is propagated to interested subscribers who can assign appropriate feed names and confirm the validity of the new feed definitions.

It is important to note that Bistro feed analyzer currently only uses filename structure to detect new data feeds or suggest refined feed definitions. In our experience with network measurement data feeds at AT&T we discovered that there is sufficient amount of metadata embedded within filenames that makes examining the content of data files unnecessary. Incorporating tools such as LEARNPADS [5] for automatic discovery of the structure of data files into the feed classification process and Bistro feed analyzer is one of the directions we are planning to take in the future.

Given that most of the feeds managed by a feed management server are not under their control, an important feature of Bistro is to perform extensive logging to track the status of all the feeds, monitor their progress (e.g., if the expected data is incomplete), detect and correct any errors, and alarm if it is unable to correct errors.

4. BISTRO FEED DISTRIBUTION

In section 2.2 we discussed major drawbacks of existing pull and push-based mechanisms employed for dissemination of real-time data feeds to subscribers, including unreliable data delivery, excessive polling overhead, lack of real-time data propagation and subscriber notification and poor resource management. We designed the Bistro feed delivery mechanism to address these challenges as well as serve as a research platform for experimenting with new delivery mechanisms and policies. In this section we will describe communication protocols we use between feed provider, managers and subscribers, notifications and trigger support, mechanisms for ensuring reliable data delivery and scheduling policies employed by the system.

4.1 Communication Protocols

In our analysis of deficiencies of existing feed delivery systems, we observed that main issue lies not with using pull or push-based data transmission, but rather with the poor communication protocols used. Feed sources producing stream of data files know when each file is generated, however this information is not communicated to feed distribution servers responsible for delivery of feeds to subscribers. Similarly, feed distribution servers have no well defined protocols to communicate to subscribers downstream that a file or batch of files has been delivered or is available for retrieval. This disconnect between feed providers and subscribers forces feed delivery systems to rely on expensive polling and introduces propagation delays unacceptable for real-time application.

To address the disconnect between the feed producers and consumers we defined a lightweight communication interface that allows feed producers to notify a Bistro server that new batch of data has been delivered and is ready for distribution to interested subscribers. We supply lightweight client software that implements this protocol and can be easily incorporated into existing software running on feed generating side or be invoked on feed manager side. In the simplest case the notifications are done on per-file basis which insures minimal propagation delays for real-time applications, such as visualization software. However, as we discussed earlier in Section 2.3 many of real-world feeds are aggregate in their nature and can be seen as stream of batches of files, each batch consisting of the data for particular time interval generated by several sources. Subscribing applications, such as streaming data warehouses, will want to wait until all of the data for particular time interval is received before triggering the update process. The Bistro communication interface allows data sources mark end-of-batch events in the output data stream to enable propagation of this information further downstream to applications. Data source specific end-of-batch markers in Bistro perform a function very similar to stream punctuations widely used by data stream management systems [12][18], and enable a variety of performance optimizations for streaming data warehouses.

Even though we believe that incorporating Bistro feed notification interface into data sources requires minimal changes, it is unrealistic to expect that all the data sources will be willing to make changes to their processes and deployed software. As we discussed earlier data sources are frequently managed by autonomous organizations and do not allow any control over the way data is being generated or distributed. Based on these conditions, the design goal for Bistro feed delivery mechanism was to be robust in the presence of autonomous sources that just push the data to feed management server without using notifications. In order to avoid expensive filesystem polling to discover newly arriving data files, the system uses concept of the *landing zones* where all data providers are depositing their files. Bistro minimizes the overhead of directory scans by immediately moving incoming files to staging directories and thus keeping the landing directories small. By using the landing zone approach for distributing network measurement data from more than one hundred non-cooperating data sources to several data warehouses, we were able to achieve sub-minute data source to application propagation delays.

Considering that non-cooperating feed sources will generally not be providing explicit punctuation marking logical ends of the batches, the Bistro feed specification language allows applications to specify the rules for generating end-of-batch notifications. One way to specify the batch size feed configuration specification is to use a fixed file count. For example, an application subscribing to a network measurement data feed composed from data from three SNMP pollers could register a trigger to be invoked whenever a new batch of three files has been delivered. This approach alone is not very robust in the presence of unreliable or dynamically changing data feeds where number of pollers producing the data for specific time interval changes from time to time. If one poller does not produce reading during particular time interval, it will not only delay the notification till a first file for the next time interval arrives, but will also generate notification in the middle of the next interval. Time-based batch specification in Bistro feed configuration language allows to specify a maximum length of the time interval a single batch can span, which is more robust in the presence of dynamic and unreliable data sources. However, it still introduces potential delays in the way notifications are delivered and can break file batches in the middle. In our experience, we found that using a combination of count and time-based batch specification works well in practice and allows application to fine-tune notification behavior. Ideally, we would like to incorporate machine learning techniques to dynamically determine end of batches events by continuously monitoring file arrival patterns. This is one of the research directions we are planning to take in the future.

Similar to communication interface between the data sources and feed management server, Bistro also defines a trigger interface to communicate with subscribers that want to invoke application-specific processing whenever new data arrives. To accommodate wide variety of subscribing applications, Bistro supports two methods of trigger invocation. The first method allows applications to specify a lightweight program (*trigger*) to be invoked by Bistro on subscriber site whenever new file or batch of files is available. For example, a streaming data warehouse could specify a trigger that will invoke the loading process for new data that will update relevant temporal data partitions. An alternative method is for applications to supply a script that will be invoked locally by the Bistro server to perform the necessary subscriber notification. Having both of these interfaces available simplifies the system's acceptance process and removes the need for changes to be made on application side. We note that even though in its current implementation Bistro is only used to push data to subscribers, there is nothing that prevents subscribers from using a hybrid push-pull approach. In this approach the data feed management server will push notification to subscribers by invoking registered trigger scripts, while applications will pull the data after relevant notifications are received at the time of their choosing.

4.2 Reliable Feed Delivery

A data feed management system is expected to provide a guarantee that every file received from a data source that matches definition of a particular feed will be delivered to all the feed's subscribers (or subscribers will be notified and given an opportunity to retrieve the file themselves). There are number of factors that make ensuring this guarantee non-trivial:

- Subscribers are not expected to be always available and can frequently go down due to software, hardware or communication failures.
- New feed subscribers can be added at any moment with the expectation that they will be receiving a full available history of the data matching feed definition.
- A feed definition can be revised at any moment with the expectation that all the files matching new definition will be disseminated to feed subscribers.
- Feed manager servers can themselves fail due to hardware or software failure and become inaccessible due to network partitioning.

In order to implement a reliable feed delivery, a feed manager server will need to manage persistent state for all of received files, feeds, subscribers and file delivery receipts. As we discussed in Section 2.1, pull-based systems offload the job of managing this state to subscribing applications, forcing them to discover all the files that have not yet been retrieved. Push-based systems such as *rsync* do not store any state instead relying on comparisons between local and subscriber's filesystems to compute a list of files that haven't been delivered. To ensure reliable feed data delivery in Bistro feed manager, we implemented a state management mechanism based on a transactional database. Every file received from data feed providers is logged in an *arrival_receipts* database along with list of feeds that the file belongs to. Additionally a separate *delivery_receipts* database is maintained that for each file stores a list of subscribers it has been delivered to. Based on the state of these two databases Bistro feed manager can always compute the content of subscriber's *delivery queues* - a list of files that have not been delivered to a particular subscriber.

In addition to managing feed arrival and delivery receipts, a feed manager must be able to monitor the state of all the subscribers, detect the failures to suspend delivery attempts as well as detect the moments subscribers come back online to resume feed delivery. In Bistro we detect subscriber failures by monitoring the errors in file transmission attempts and eventually flagging subscribers as being offline. Transmissions are periodically retried to check if failed subscriber was brought back online after which moment the content of delivery queue is recomputed and undelivered files are scheduled to be backfilled to subscriber's host.

Given the impossibility of storing a potentially infinite history of the files generated by every feed provider, every Bistro server maintains a limited time window of data and regularly expunges files that fall outside of the window. The size of the stored time windows will vary based on available storage capacity and subscriber requirements. A typical new subscriber is more interested in real-time data and will generally have no problems with restricted amount of historical data available on particular Bistro server. However, in some cases a subscriber needs to perform a long-term data analysis over wide range of historical data large then the time available at feed management server. In order to serve the requirements of such subscribers, the Bistro feed manager implements an archival mechanism by maintaining a set of special *archiver* nodes that are responsible for storing long-term feed history and optionally undo/redo logs of delivery receipt database on tertiary storage. The archival mechanism also provides additional level of resilience for catastrophic storage-failures of Bistro server itself when there is a danger of potential loss of short-term data feed history and delivery receipt databases.

4.3 Feed Delivery Scheduling

An important requirement for feed management systems is to guarantee that feed files are propagated from data sources to subscribing applications with well-defined tardiness. Real-time applications such as visualization systems, network alerts monitors and intrusion detection systems are particularly sensitive to propagation delays introduced by feed delivery infrastructure. Given limited processing resources and storage bandwidth, as well as restricted bandwidth of network links connecting feed manager server with the subscribers, a real-time scheduling algorithm must be employed to ensure that delivery deadlines are met. A wide variety of real-time scheduling algorithms has been studied in literature, including Max Benefit [6], Earliest Deadline First (EDF) [10], Prioritized EDF (EDF-P) [6] and Rate Monotonic (RT) [15] algorithms. However, the problem of scheduling the delivery of data feeds presents a number of unique challenges:

Several constrained resources. A feed management server has a number of constrained system resources including CPU (number and processing power of individual cores), memory, storage system bandwidth, network bandwidth available to each of the subscribers and performance capabilities of individual subscribers. Most known real-time scheduling algorithms do not work well in a system with several constrained resources.

Data backlogs due to subscriber unavailability. Subscribers are not guaranteed to be always available and any feed management system serving large numbers of subscribers should expect fraction of subscribers to be offline. Prolonged periods of unavailability lead to significant backlogs forming on data feed manager that will need to be backfilled when a subscriber comes back online. Propagating backlogs to previously offline subscribers can lead to starvation of delay-sensitive feed subscribers.

Note that there are two possible backfill strategies that a Data Feed Management system could employ. One strategy is to guarantee that data feeds will be delivered in the same order they were received by DFMS. However, this approach sacrifices the real-time delivery guarantees that are important for many applications. Alternatively, we can relax the requirement for in-order feed delivery and deliver new data in real-time concurrently with backfilling of missed historical data. Given Bistro focus on real-time applications we implemented the latter strategy.

High subscriber heterogeneity. Different subscribers can vary widely in their processing power, storage bandwidth and network connectivity. As a result, the delivery of the same file to different subscribers can take vastly different times. A good scheduling algorithm will need to ensure that slow and overloaded subscribers do not starve more responsive ones by taking disproportionally high share of server resources.

Bistro feed manager addresses these scheduling challenges by partitioning subscribers into several levels based on their overall responsiveness. Each partition is allocated fixed amount of system resources (cpu cores) and is free to utilize its own scheduling algorithm to allocate these resources. Given homogenous nature of subscribers in each partition, intra-partition scheduling is much easier and many scheduling algorithms including EDF work very well. Similar approach of partitioning tasks into homogenous groups each running local scheduling algorithm such as EDF was shown to work well for real-time update propagation in streaming data warehouses [6]. We also use a number of heuristics to optimize the temporal and spatial locality of storage accesses for different subscribers, for example by making sure the delivery of a file to several subscribers within a group is performed concurrently whenever possible. Current implementation of Bistro feed manager only supports fixed small number of scheduling groups and does not support dynamic migration of subscriber from one group to another based on observed runtime behavior. Incorporating dynamic subscriber partitioning into Bistro scheduling algorithm is a research direction we are planning to explore in the future work.

5. BISTRO FEED ANALYZER

Whenever a new file arrives into one of the Bistro landing directories, they need to be classified as belonging to one or more of data feeds registered in the system. The accuracy of the classification process is critical for all the applications that subscribe to data feeds managed by the system. Files incorrectly classified as a part of the feed or incorrectly excluded from the feed can go unnoticed for a long time, introducing serious data quality issues. Even when missing or extraneous data files are detected by the subscribing application and traced back to incorrect feed definition, a process during which all subscribers will need to agree on a new feed definition may take considerable amount of time. Furthermore, due to feed evolution over time, a definition that used to be correct may no longer reflect a new feed composition. The job of *Bistro Feed Analyzer* is to proactively monitor a stream of incoming data files and classification decisions made by the system in order to detect potential misclassifications and generate alternative feed definitions for subscribers' consideration. In this section, we will describe three main components of the feed analyzer: new feed discovery, refining feed definitions to eliminate false negatives, and finally correcting false positives in feed classification.

5.1 New Feed Discovery

Many of real-world data feeds that Bistro feed manager is tasked to manage are highly aggregate in their nature and in some cases contain more than a hundred individual subfeeds coming from diverse data sources. Usually not all of the individual subfeeds are known to Bistro administrators or potential subscribers; in some extreme cases we observed feeds with more than half of the files falling into "unknown feed" category. As we briefly discussed in Section 2, manual feed discovery is very laborious processes with high probability of incorrect feed definition being generated. Bistro feed discovery module automates the process of discovery of new feeds by generating a list of suggested feed definitions to be reviewed by potential subscribers.

Generally, the Bistro feed manager treats data feeds as black boxes and is not given any information about the formats of the data files that it needs to manage. However, there is frequently a wealth of information that is encoded inside the filenames themselves which provides sufficient information to make a classification decision. Consider for example the following set of files:

MEMORY_POLLER1_2010092504_51.csv.gz,
CPU_POLL1_201009250502.txt,
MEMORY_POLLER2_2010092504_59.csv.gz,
MEMORY_POLLER1_2010092509_58.csv.gz,
CPU_POLL2_201009250503.txt,
MEMORY_POLLER2_2010092510_02.csv.gz,
CPU_POLL2_201009251001.txt,
CPU_POLL2_201009250959.txt

By visual inspection we can identify two classes of files, one of the form *MEMORY_POLLERid_YYYYMMDDHH_MM.csv.gz* and other of the form *CPU_POLLid_YYYYMMDDHHMM.txt*. We can also with reasonable confidence identify timestamp fields in both classes of files and observe that *id* field takes values from the domain {1, 2}. Finally, by looking at file arrival/creation timestamps we can conclude that both classes of files should expect to see a new file generated every 5 minutes from each of the pollers. Generally, we cannot automatically determine if both of the classes of files belong to the same feed or not since any feed can be defined using arbitrary grouping of smaller subfeeds. Instead Bistro feed analyzer identifies homogeneous group of files called *atomic feeds*. Intuitively, an atomic feed is produced by a single data generating software using consistent file naming convention.

An atomic feed is defined as a sequence of files sharing the same structure of the filename, i.e. the same sequence of tokens or fields. General problem of string tokenization is very hard given than some filenames use fixed-length fields of unknown length instead of traditional separators. Bistro uses a collection of heuristics to identify fixed-length field boundaries, including detecting changes between alphabetic and numeric characters as well as recognizing common field formats (dates, numbers, ip addresses). For each field in a filename Bistro computes its field types and corresponding domains, e.g fixed-value string, categorical variable, integer, timestamp, etc.

Once the field structure of individual filenames has been identified, sequences of filenames sharing the same set of fields and field domains are clustered together to form atomic feed definitions. By analyzing sequences of files belonging to the same feed, Bistro determines files arrival and batching patterns to be included in feed definitions. Currently, there is no support for automatic grouping of atomic feeds into larger feed groups; instead Bistro relies on human domain experts to define complex group hierarchies based on the output of feed discovery module. Developing tools for automatic grouping of related or structurally similar atomic feeds into more complex logical feed groups is one of the research directions we are planning to undertake in the future.

5.2 False Negatives in File Classification

An important property of good feed specification is robustness to future changes in feed compositions. Consider a feed containing the following files:
MEMORY_poller1_20100925.gz,
MEMORY_poller2_20100925.gz,
MEMORY_poller1_20100926.gz,
MEMORY_poller1_20100926.gz
A reasonable feed definition would be *MEMORY_poller%i_%Y%m%d.gz* which specifies that all the filenames will start with *MEMORY_poller* followed by an integer and then a timestamp. Small changes in the feed file naming convention on the source side (such as capitalizing *p* in a string *poller*) will make system fail to produce a match for files like *MEMORY_Poller1_20100926.gz*. Even though the feed definition can be easily adjusted to include new files once precise nature of the problem has been identified, detecting and reporting it automatically presents a major challenge.

Intuitively an unmatched filename *n* can be considered a potential false negative for existing feed *F* if there is a high similarity between *n* and files that match *F*. One obvious choice of similarity metric is a string edit distance between a string and feed pattern. More specifically, the minimum number of insertions, deletions or substitutions that need to be made to a string to make it match existing feed pattern. Even though this definition of file to feed similarity would work for detecting false negatives in the above example, our experience shows that false negatives can exhibit a very large edit distance. Consider the following feed pattern *TRAP__%Y%m%d_DCTAGN_klpi.txt* and potential false negative file

TRAP_2010030817_UVIPTV-PER-BAN-DSPS-IPTV_MOM-rcsntxsqlcv122_9234SEC_klpi.txt

Even though intuitively this file is highly similar to a feed pattern above, an edit distance of 51 significantly exceeds the length of the common parts of the filename.

Alternative approach for detecting false negatives in filename classification is to generate generalized patterns for an unmatched file and use a similarity between this pattern and the patterns in existing feed definition. This is the approach that is currently being used by Bistro feed analyzer. We run feed discovery tools to generate generalized patterns describing unmatched files and then use pattern similarity to generate a list of possible false negatives. An added benefit of Bistro approach is in significant reduction in the number of warning messages generated for each suspected match, since a warning is only generated once for each generalized file pattern.

5.3 False Positives in File Classification

Most commonly false positives in file classification are identified during ETL process on subscriber side. Once extraneous data are detected, a subscriber would work with feed manager administrators to modify the original feed specification to exclude unwanted files. However, if ETL process does not trigger any alarms bad data can propagate quietly through subscriber application logic leading to hard to detect data quality issues. Ideally, we would like to be able to automatically identify potential false positives and correct initial feed definitions without any subscriber intervention.

In general case the task of identification of extraneous files is impossible since a feed manager has no reliable way to tell whether a given file was meant to be part of a certain feed or not. A feed definition can potentially group arbitrary sets of files with no commonality in either file structure or arrival patterns. Instead of directly identifying false positives Bistro feed analyzer explores the stream of files matching existing feed definition and identifies all the contained *atomic feeds* using new feed discovery algorithm described in Section 5.1. In addition to clustering filenames into atomic feeds the system identifies and marks outliers that do not share filename structure with the rest of the matching files. A list of atomic feed definitions is then forwarded to all the feed subscribers giving them the opportunity to verify that all the individual subfeeds included in currently installed feed definition are there intentionally

Bistro's proactive approach to identifying potential errors in feed definitions gives subscribers the opportunity to make corrections before the bad quality data propagates through the application logic.

6. RELATED WORK

Publish-subscribe systems [9] have been extensively studied in the literature, covering a wide range of topics: languages for expressing subscriptions [8], matching algorithms [16], security [19], real-time processing [11], multicast protocols for subscriber notifications [1], and many other aspects. A large number of systems have been built both as research prototypes and as commercial systems. A key difference between publish-subscribe systems and Bistro feed manager is that they focus on delivery of short messages rather than large-volume data feeds. Publication to subscriber mapping in the context of data feed management is also very different since no assumption is being made that mapping predicates are known in advance or that they will remain correct during the system lifetime.

A content-distribution system for publish-subscribe services proposed in [3] addresses a problem of distributing large-size content by pushing matching content closer to subscribers and caching it based on their access patterns. A general assumption here is that a matching content will be retrieved on demand, so the main objective of the system is to minimize the cost of data retrieval. In a contrast, the Bistro feed management system is designed for immediate propagation of the feed data with real-time constraints.

The problem of automatic learning of filename structure to discover feed definitions is closely related to a problem of discovering data file structure for ad-hoc data analysis studied in the context of PADS project [4]. A LEARNPADS system [5] automatically infers data file definitions based on a sample of data records. An incremental version of LEARNPADS [20] allows revising initial data definitions based on new data that becomes available. Compared to Bistro, both incremental and regular versions of the system treat input as sets rather than sequences of records and do not take advantage of temporal information present in the input stream.

7. CONCLUSIONS

Large organizations with diverse operations generate a wide variety of data feeds from operations at geographically distributed locations. Many critical business applications including continuous business analytics systems, complex-event processing systems, traditional and streaming data warehouses and data visualization systems rely on efficient real-time delivery of these data feeds. Managing high-volume real-time data feeds in robust and efficient manner requires solving a large number of technical problems: feed discovery, real-time feed delivery, data transmission scheduling, feed analysis for quality monitoring and many others. In this paper we analyze all the major challenges in data feed management and show deficiency of currently used tools.

We describe the *Bistro* data feed management system that we developed to handle the industrial-scale data feed management problems we encountered while building data systems for AT&T. Bistro servers currently manage over 100 data feeds, delivering up to 300 gigabytes of data per day to a number of customers in real-time, while also providing real-time delivery alerts. We demonstrated how Bistro design choices address the feed management challenges and presented the avenues for improvements and future research directions.

8. REFERENCES

[1] G. Banavar, T. D. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, D. C. Sturman. An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems. *In Proc ICDCS* 1999: 262-272.

[2] A. Carzaniga, D. Rosenblum, A. Wolf. Achieving scalability and expressiveness in an Internet-scale event notification service. *Proc. ACM PODC 00.* ACM Press, New York, NY. 2000.

[3] M. Chen, A. LaPaugh, and J. P. Singh. Content distribution for publish/subscribe services. *In Proc International Middleware Conference*, 2003.

[4] K. Fisher and R. Gruber. PADS: A domain specific language for processing ad hoc data. *In Proc PLDI*, pages 295–304, June 2005.

[5] K. Fisher, D. Walker, K. Q. Zhu. LearnPADS: automatic tool generation from ad hoc data. *In Proc ACM SIGMOD 2008*: 1299-1302.

[6] L. Golab, T. Johnson, V. Shkapenyuk: Scheduling Updates in a Real-Time Stream Warehouse. *Proc ICDE 2009*: 1207-1210

[7] L. Golab, T. Johnson, J. Seidel, V. Shkapenyuk. Stream Warehousing with DataDepot. *Proc. ACM SIGMOD 2009*.

[8] R. E. Gruber, B. Krishnamurthy, E. Panagos. High-level constructs in the READY event notification system. *In Proc ACM SIGOPS European Workshop 1998*: 195-202.

[9] H. Jacobsen. Publish/Subscribe. *Encyclopedia of Database Systems 2009*: 2208-2211.

[10] J.R. Jackson. Scheduling a production line to minimize maximum tardiness. Management Science Research Project 43, University of California, Los Angeles, 1955.

[11] Z. Jerzak, R. Fach, C. Fetzer. Fail-Aware Publish/Subscribe. *In Proc NCA 2007*: 113-125.

[12] T. Johnson, S. Muthukrishnan, V. Shkapenyuk, O. Spatscheck: A Heartbeat Mechanism and Its Application in Gigascope. *In Proc VLDB 2005*: 1079-1088.

[13] C. R. Kalmanek, Z. Ge, S. Lee, C. Lund, D. Pei, J. Seidel, J. van der Merwe, and J. Yates. Darkstar: Using exploratory data mining to raise the bar on network reliability and performance. *In Proc Workshop on Design of Reliable Communication Networks*, 2009.

[14] S. Krishnamurthy, M. J. Franklin, J. Davis, D. Farina, P. Golovko, A. Li, N. Thombre: Continuous analytics over discontinuous streams. *In Proc SIGMOD Conference 2010*: 1081-1092.

[15] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in real-time environment. *Journal of the ACM*, vol. 20(1), pp. 46-61, 1973.

[16] J. Pereira, F. Fabret, F. Llirbat, D. Shasha. Efficient Matching for Web-Based Publish/Subscribe Systems. *In Proc CoopIS 2000*: 162-173.

[17] rsync documentation. http://www.manpagez.com/man/1/rsync/

[18] P. A. Tucker, D. Maier, T. Sheard, L. Fegaras. Exploiting Punctuation Semantics in Continuous Data Streams. *In Proc IEEE Transactions on Knowledge and Data Engineering*, v.15 n.3, p.555-568, March 2003.

[19] N. Uramoto, H. Maruyama. InfoBus Repeater: A Secure and Distributed Publish/Subscribe Middleware. *In Proc ICPP Workshops 1999.*

[20] K. Q. Zhu, K. Fisher, D. Walker. Incremental learning of system log formats. *In Proc Operating Systems Review* 44(1): 85-90 (2010).