The “Big Data” Ecosystem at LinkedIn

Roshan Sumbaly, Jay Kreps, and Sam Shah
LinkedIn

ABSTRACT

The use of large-scale data mining and machine learning has proliferated through the adoption of technologies such as Hadoop, with its simple programming semantics and rich and active ecosystem. This paper presents LinkedIn’s Hadoop-based analytics stack, which allows data scientists and machine learning researchers to extract insights and build product features from massive amounts of data. In particular, we present our solutions to the “last mile” issues in providing a rich developer ecosystem. This includes easy ingress from and egress to online systems, and managing workflows as production processes. A key characteristic of our solution is that these distributed system concerns are completely abstracted away from researchers. For example, deploying data back into the online system is simply a 1-line Pig command that a data scientist can add to the end of their script. We also present case studies on how this ecosystem is used to solve problems ranging from recommendations to news feed updates to email digesting to descriptive analytical dashboards for our members.

Categories and Subject Descriptors: H.2.4 [Database Management]: Systems; H.2.8 [Database Management]: Database Applications
General Terms: Design, Management
Keywords: big data, hadoop, data mining, machine learning, data pipeline, offline processing

1. INTRODUCTION

The proliferation of data and the availability and affordability of large-scale data processing systems has transformed data mining and machine learning into a core production use case, especially in the consumer web space. For instance, many social networking and e-commerce web sites provide derived data features, which usually consist of some data mining application offering insights to the user.

A typical example of this kind of feature is collaborative filtering, which showcases relationships between pairs of items based on the wisdom of the crowd (“people who did *this* also did *that*”). This technique is used by several organizations such as Amazon [20], YouTube [5], and LinkedIn in various recommendation capacities. At LinkedIn, the largest professional online social network with over 200 million members, collaborative filtering is used for people, job, company, group, and other recommendations (Figure 2b) and is one of the principal components of engagement.

Other such derived data applications at LinkedIn include “People You May Know,” a link prediction system attempting to find other users you might know on the social network (Figure 2a); ad targeting; group and job recommendations; news feed updates (Figure 3a); email digesting (Figure 3b); analytical dashboards (Figure 4a); and others. As a smaller example, LinkedIn displays “related searches” (Figure 2d) on the search results page. This feature allows users to refine their searches or explore variants by pivoting to alternate queries from their original search [27]. There are over a hundred of these derived data applications on the site.

These applications are largely enabled by Hadoop [34], the open-source implementation of MapReduce [6]. Among Hadoop’s advantages are its horizontal scalability, fault tolerance, and multitenancy: the ability to reliably process petabytes of data on thousands of commodity machines. More importantly, part of Hadoop’s success is its relatively easy-to-program semantics and its extremely rich and active ecosystem. For example, Pig [23] provides a high-level dataflow language; Hive [33] provides a dialect of SQL; and libraries such as Mahout [24] provide machine learning primitives.

This rich development environment allows machine learning researchers and data scientists—individuals with modest software development skills and little distributed systems knowledge—to extract insights and build models without a heavy investment of software developers. This is important as it decreases overall development costs by permitting researchers to iterate quickly on their own models and it also elides the need for knowledge transfer between research and engineering, a common bottleneck.

While the Hadoop ecosystem eases development and scaling of these analytic workloads, to truly allow researchers to “productionize” their work, ingress and egress from the Hadoop system must be similarly easy and efficient, a presently elusive goal. This data integration problem is frequently cited as one of the most difficult issues facing data practitioners [13]. At LinkedIn, we have tried to solve these “last mile” issues by providing an environment where researchers have complete, structured, and documented data available, and where they can publish the results of their algorithms without difficulty. Thereafter, application developers can read these results and build an experience for the user. A key characteristic of our solution is that large-scale, multi-data center data deployment is a 1-line command, which provides seamless integration into existing Hadoop infrastructure and allows the researcher to be agnostic to distributed system concerns.

This paper describes the systems that engender effortless ingress and egress out of the Hadoop system and presents case studies of how data mining applications are built at LinkedIn. Data flows into the system from Kafka [15], LinkedIn’s publish-subscribe system.

Data integrity from this pipeline is ensured through stringent schema validation and additional monitoring.

This ingress is followed by a series of Hadoop jobs, known as a workflow, which process this data to provide some additional value—usually some predictive analytical application. Azkaban, a workflow scheduler, eases construction of these workflows, managing and monitoring their execution. Once results are computed, they must be delivered to end users.

For egress, we have found three main vehicles of transport are necessary. The primary mechanism used by approximately 70% of applications is key-value access. Here, the algorithm delivers results to a key-value database that an application developer can integrate with. For example, for a collaborative filtering application, the key is the identifier of the entity and the value is a list of identifiers to recommend. When key-value access is selected, the system builds the data and index files on Hadoop, using the elastic resources of the cluster to bulk-load Voldemort [31], LinkedIn's key-value store.

The second transport mechanism used by around 20% of applications is stream-oriented access. The results of the algorithm are published as an incremental change log stream that can be processed by the application. The processing of this change log may be used to populate application-specific data structures. For example, the LinkedIn "news feed" provides a member with updates on their network, company, and industry. The stream system permits any application to inject updates into a member's stream. An analytics application (e.g., "What are the hot companies in your network?") runs its computation as a series of Hadoop jobs and publishes its feed of new updates to the stream service at the end of its computation.

The final transport mechanism is multidimensional or OLAP access. Here, the output is a descriptive statistics application, where filtering and aggregation is used for cube materialization. This output feeds frontend dashboards to support roll ups and drill downs on multidimensional data. For example, LinkedIn provides analytics across various features of a member's profile view.

Given the high velocity of feature development and the difficulty in accurately gauging capacity needs, these systems are all horizontally scalable. These systems are run as a multitenant service where no real stringent capacity planning needs to be done: rebalancing data is a relatively cheap operation, engendering rapid capacity changes as needs arise.

The main contribution of this work is an end-to-end description and historical reasoning of a large-scale machine learning environment and the constituent systems that provide smooth, effortless integration for researchers. These systems have been in use for over three years at LinkedIn and have been widely successful in allowing researchers to productionize their work. In fact, none of the machine learning teams at LinkedIn have any application developers or systems programmers whose responsibility is productionization; the researchers do this themselves.

The components described in this paper are open source and freely available under the Apache 2.0 license.

2. RELATED WORK

There is little literature available on productionizing machine learning workflows. Twitter has developed a Pig [23] extension for stochastic gradient descent ensembles for machine learning [19] that it uses internally. This provides ease of model development, but the end-to-end production pipeline—which is regularly and reliably delivering data updates to predictive analytical applications—is not described. Similarly, Facebook uses Hive [33] for its data analytics and warehousing platform [32], but little is known on productionizing its machine learning applications.

In terms of ingress, ETL has been studied extensively. For Hadoop ingress, in Lee et al. [16], the authors describe Twitter's transport mechanisms and data layout for Hadoop, which uses Scribe [33]. Scribe was originally developed at Facebook and forms the basis for their real-time data pipeline [3]. Yahoo has a similar system for log ingestion, called Chukwa [26]. Instead of this approach, we have developed Kafka [15], a low-latency publish-subscribe system that unifies both near-line and offline use cases [11]. These Hadoop log aggregation tools support a push model where the broker forwards data to consumers. In a near-line scenario, a pull model is more suitable for scalability as it allows consumers to retrieve messages at their maximum rate without being flooded, and it also permits easy rewind and seeking of the message stream.

In terms of egress from batch-oriented systems like Hadoop, MapReduce [6] has been used for offline index construction in various search systems [21]. These search layers trigger builds on Hadoop to generate indexes, and on completion, pull the indexes to serve search requests. This approach has also been extended to various databases. Konstantinou et al. [14] and Barbuzzi et al. [2] suggest building HFiles offline in Hadoop, then shipping them to HBase [9], an open source database modeled after BigTable [4]. In Silberstein et al. [30], Hadoop is used to batch insert data into PNUTS [29], Yahoo!'s distributed key-value solution, in the reduce phase.

OLAP is a well studied problem in data warehousing, but there is little related work on using MapReduce for cubing and on serving queries in the request/response path of a website. MR-Cube [22] efficiently materializes cubes using the parallelism of a MapReduce framework, but does not provide a query engine.

None of these egress systems explore the data pipeline or describe an end-to-end pipeline for data mining applications.

3. THE ECOSYSTEM

Figure 1 shows the architecture of LinkedIn's data pipeline. Member and activity data generated by the online portion of the website flows into our offline systems for building various derived datasets, which is then pushed back into the online serving side. As early adopters of the Hadoop stack, LinkedIn has been using HDFS [28], the distributed filesystem for Hadoop, as the sink for all this data. HDFS then acts as the input for further processing for data features.

The data coming into HDFS can be classified into two categories: activity data (❷ in Figure 1) and core database snapshots (❸ in Figure 1). The database change-logs are periodically compacted into timestamped snapshots for easy access. The activity data consists of streaming events generated by the services handling requests on LinkedIn. For example, a member viewing another profile would count as a single event. Events are grouped into semantic *topics* and transported by LinkedIn's publish-subscribe system, Kafka. These topics are eventually pulled into a hierarchical directory structure onto HDFS. We discuss this pipeline in more detail in Section 4.

Once data is available into an ETL HDFS instance, it is then replicated to two Hadoop instances, one for development and one for production. Here, researchers and data scientists define a *workflow* of jobs to compute some value-added derived data. We use Azkaban, LinkedIn's open source scheduler, to manage these workflows. Azkaban maintains a global view of workflows, which aids monitoring, resource locking, and log collection. A job in the context of Azkaban is defined in an easy key-value configuration file format with dependent jobs mentioned as part of the configuration. Azkaban is a general purpose execution framework and supports diverse job types such as native MapReduce, Pig, Hive, shell scripts, and others. LinkedIn production workflows are predominantly Pig, though native MapReduce is sometimes used for performance rea-

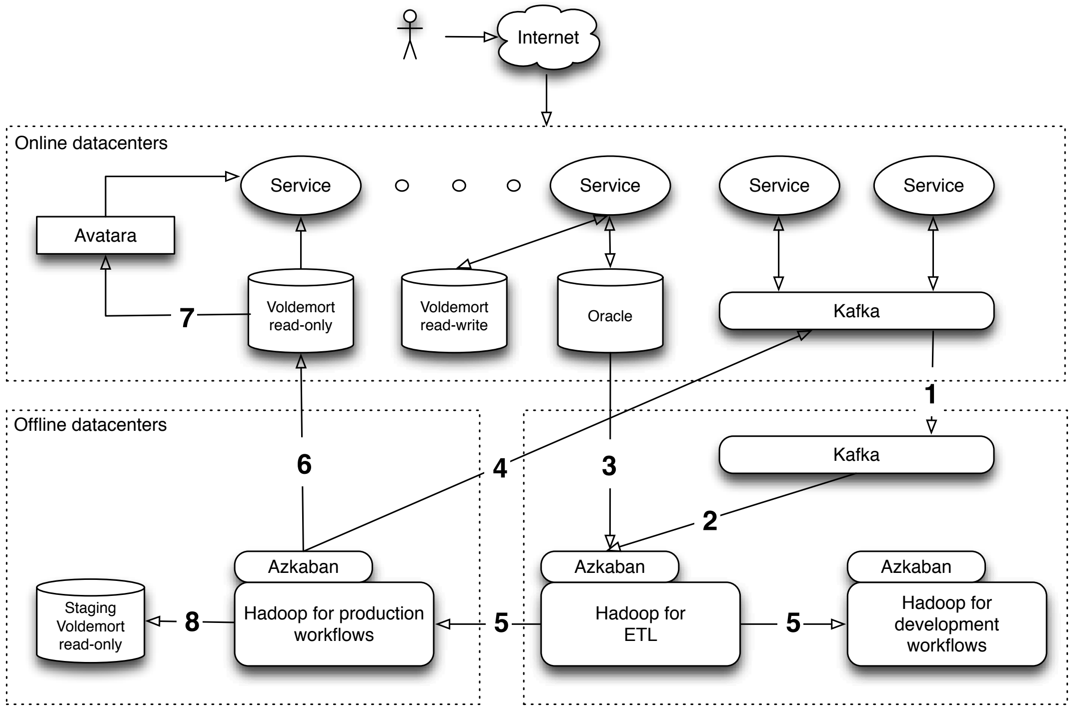


Figure 1: Architecture of the data pipeline at LinkedIn. The online data center serves user requests while the offline data centers host Hadoop for further offline processing

sons. To ease the development process of prototyping with our data, user-defined functions are shared in a library called DataFu, which is also open source. We discuss the deployment and usage of Azkaban in Section 5.

Finally, the output of these workflows feeds back into our online serving systems. Outlets of streams from Kafka (❹ in Figure 1), key-value databases using Voldemort (❻ in Figure 1) and multidimensional OLAP cubes using Avatara (❼ in Figure 1) have a simple single-line Pig statement at the end of a workflow.

Before deploying data back to the online serving systems, some workflows may choose to push data to staging clusters (❽ in Figure 1) with extra debugging information. This facilitates easy debugging through “explain” commands to understand the nature of results and the output of models. Section 6 covers the individual egress systems as well as how they integrate into the complete application workflow.

4. INGRESS

Data loaded into Hadoop comes in two forms: database and event data. Database data includes information about users, companies, connections, and other primary site data. Event data consists of a stream of immutable activities or occurrences. Examples of event data include logs of page views being served, search queries, and clicks.

The primary challenge in supporting a healthy data ecosystem is providing infrastructure that can make all this data available without manual intervention or processing. There are several difficulties in achieving this. First, datasets are large so all data loading must scale horizontally. Second, data is diverse: LinkedIn maintains well over a thousand different datasets that must be continuously replicated. Third, data schemas are evolving as the functionality of the site itself changes rather rapidly. This is particularly challenging for data that is retained over time, as we must ensure new data is compatible with older data. Finally, because data collection interacts with thousands of machines in multiple data centers, each with various operational considerations, monitoring and validating data quality is of utmost importance. If a dataset is incomplete or incorrect, all processing on it naturally produces incorrect results.

For simplicity, we will focus on the activity data pipeline, though the database data goes through a similar process. LinkedIn has developed a system called Kafka [15] as the infrastructure for all activity data. Kafka is a distributed publish-subscribe system [8] that persists messages in a write-ahead log, partitioned and distributed over multiple brokers. It allows data publishers to add records to a log where they are retained for consumers that may read them at their own pace. Each of these logs is referred to as a *topic*. An example of this might be collecting data about searches being performed. The search service would produce these records and publish them to a topic named “SearchQueries” where any number of subscribers might read these messages.

All Kafka topics support multiple subscribers as it is common to have many different subscribing systems. Because many of these feeds are too large to be handled by a single machine, Kafka supports distributing data consumption within each of these subscribing systems. A logical consumer can be spread over a group of machines so that each message in the feed is delivered to one of these machines. Zookeeper [12], a highly-available distributed synchronization service, is used for these groups to select a consumer machine for every Kafka topic partition and redistribute this load when the number of consumers changes.

Activity data is very high in volume—several orders of magnitude larger than database data—and Kafka contains a number of optimizations necessary to cope with these volumes. The persistence layer is optimized to allow for fast linear reads and writes; data is batched end-to-end to avoid small I/O operations; and the network layer is optimized to zero-copy data transfer.

A critical part of making data available is that topics, consumers, brokers, and producers can all be added transparently without manual configuration or restart. This allows new data streams to be added and scaled independent of one another.

4.1 Data Evolution

Data evolution is a critical aspect of the ecosystem. In early versions of our data load processes, we did not directly handle the evolution of data schemas, and as a result each such evolution required manual intervention to issue schema changes. This is a common problem for data warehouse ETL. There are two common solutions. The first is to simply load data streams in whatever form they appear, such as files of unstructured data. The problem with this is that jobs that process these streams have no guarantees on what may change in their feed, and it can be very difficult to reason about whether a particular change in the producing application will break any consumer of that data. The more traditional approach in enterprise data warehousing is to manually map the source data into a stable, well-thought-out schema and perform whatever transformations are necessary to support this. The major disadvantage of the latter approach is that it is extremely difficult to manage: thousands of data sources means thousands of data transformations. Unsurprisingly, having a central team responsible for all of these transformations becomes a bottleneck.

Rather than loading unstructured data, our solution retains the same structure throughout our data pipeline and enforces compatibility and other correctness conventions on changes to this structure. To support this, we maintain a schema with each topic in a single consolidated *schema registry*. If data is published to a topic with an incompatible schema, it is flatly rejected; if it is published with a new backwards compatible schema, it evolves automatically. This check is done both at compile and run time to help flush out these kinds of incompatibilities early. Each schema also goes through a review process to help ensure consistency with the rest of the activity data model. LinkedIn has standardized on Apache Avro as its serialization format.

4.2 Hadoop Load

The activity data generated and stored on Kafka brokers is pulled into Hadoop using a map-only job that runs every ten minutes on a dedicated ETL Hadoop cluster as a part of an ETL Azkaban workflow (❷ in Figure 1). This job first reads the Kafka log offsets for every topic from a previous run and checks for any new topics that were created. It then starts a fixed number of mapper tasks, distributes the partitions evenly, pulls data into HDFS partition files, and finally registers it with LinkedIn's various systems (e.g., register with Hive metastore for SQL queries). By load balancing the partitions within a fixed number of tasks instead of using single tasks per partition, the overhead of starting and stopping tasks for low-volume topics is avoided. Also, automatically picking up new topics helps engineers get their data into Hadoop for analysis quickly. After data is available in the ETL cluster, this new data is copied to the production and development Hadoop clusters (❺ in Figure 1). The use of an ETL cluster saves processing costs by performing extraction and transformation only once.

Besides the frequent job, the ETL workflow also runs an aggregator job every day to combine and dedup data saved throughout the day into another HDFS location and run predefined retention policies on a per topic basis. This combining and cleanup prevents having many small files, which reduces HDFS NameNode memory pressure (a common bottleneck) and improves MapReduce performance. The final directory layout on HDFS is shown below. We maintain the history of all topics since creation on a daily basis, while keeping only the last few days of data at a ten minute granularity.

```
/data/<topic1>/day/2012/03/11/[*.avro]
...
/data/<topic1>/minute/2012/03/11/23/[*.avro]
...
```

At LinkedIn, we maintain two Kafka clusters for activity data: a primary cluster that is used for production and consumption by online services, and a secondary cluster that is a clone of the primary. The secondary cluster is used for offline prototyping and data loading into Hadoop. The two clusters are kept in sync via a mirroring process (❶ in Figure 1) that is supported by Kafka. In this process, every broker node in the secondary cluster acts as a consumer to retrieve and store batched compressed data from the primary cluster. The primary cluster also supports topics with producers from Hadoop (❹ from Figure 1), as further explained in Section 6.2.

As of writing, the Kafka clusters maintain over 100 TB of compressed data for approximately 300 topics. They handle more than 15 billion message writes each day with a sustained peak of over 200 thousand messages per second. Similarly, they support dozens of subscribers and delivers more than 55 billion messages each day.

4.3 Monitoring

The flow of data between distributed systems and between geographically distributed data centers for a diverse set of data sets makes diagnosing problems automatically of the utmost importance. To achieve this, each step in the flow—producers, brokers, replica brokers, consumers, and Hadoop clusters—all publish an audit trail that allows assessing correctness and latency for each data stream and each tier in the data pipeline. This audit data consists of the topic, the machine name, the logical processing tier the machine belongs to, a predetermined time window, and the number of events seen by a particular machine in that window. By aggregating this audit data, the system can check that all events published reached all consumers. This audit is done continuously and alerts if completeness is not reached in a pre-determined time.

If a researcher or data scientist requires data that is currently not available, they only need to talk to the engineering team responsible for the feature to register an appropriate schema and make a straightforward code change to fire said event. Thereafter, data flows automatically and reliably to all offline systems: the data scientist and the application engineer are completely agnostic to the underlying process.

For more details on Kafka and LinkedIn's ingress mechanism, the reader is directed to Goodhope et al. [11].

5. WORKFLOWS

The data stored on HDFS is processed by numerous chained MapReduce jobs that form a *workflow*, a directed acyclic graph of dependencies. Workflows are built using a variety of available tools for the Hadoop ecosystem: Hive, Pig, and native MapReduce are the three primary processing interfaces. To support data interoperability, Avro is used as the serialization format. Appropriate loaders for Avro are available for many tools and if a loader is not available, it is usually not difficult to write one.

Over time, we noticed workflows implementing common functionality, which could easily be extracted out as a library. For example, most workflows only require reading subsets of data partitioned by time. Because the Hadoop-Kafka data pull job places the data into time-partitioned folders, it became easy to provide wrappers around input and output for this purpose. In the case of Hive, we create partitions for every event data topic during the data pull, thereby allowing users to run queries within partitions as follows:

```
SELECT count(1) FROM SearchQueryEvent
   WHERE datepartition='2012-03-11-00';
```

Similarly for Pig and native MapReduce, a wrapper helps restrict data being read to a certain time range based on parameters specified. Internally, if the data for the latest day is incomplete, the system automatically falls back to finer granularity data (hourly or 10 minute granularity). There is also a wrapper around storage formats to ease writing of data.

These workflows can get fairly complex; it is not unusual to see workflows of 50–100 jobs. To manage these workflows, LinkedIn uses Azkaban, an open source workflow scheduler. Azkaban supports a diverse set of job types, and a collection of these jobs form a workflow. Each job runs as an individual process and can be chained together to create dependency graphs. Configuration and dependencies for jobs are maintained as files of simple key-value pairs. Through a user interface, researchers deploy their workflows to a particular Azkaban instance.

Researchers can also visualize and manipulate these dependency graphs in the Azkaban user interface. Azkaban allows scheduling of this graph (or subgraph) while maintaining logs and statistics around previous executions. The system provides monitoring and restart capabilities for regularly running workflows. If a job should fail, configurable alerts are triggered and once the problem is remedied, only the failed subgraph needs to be restarted. For example, if a pipeline of 100 jobs should fail at job 85, only successive jobs from job 85 would be restarted. Azkaban also provides resource locking, though this feature is often not used as most of our data is append only.

The typical construction of a production workflow is as follows. A researcher typically first starts experimenting with the data through a script, trying to massage it into a form they need. If this is a machine learning application, data cleaning and feature engineering typically require the most time [13]. Each feature becomes an individual Azkaban job followed by a join of the output of these jobs into a feature vector. The researcher can simply set the predecessors of the feature aggregation job to be each individual feature job. These features can be trained into a model and we have several packaged jobs to do this. As the researcher adds or modifies these features, they configure Azkaban to only execute the changed jobs and their dependencies; Azkaban handles the rest. As workflows are iterated upon and mature, they naturally become more complex and Azkaban becomes more useful.

LinkedIn maintains three Azkaban instances, one corresponding to each of our Hadoop environments. Azkaban on the ETL grid manages the Hadoop load (Section 4.2) and is completely hidden from users of the Hadoop ecosystem. For the development and production environments, a researcher first deploys their workflows on the developer Azkaban instance to test the output of their algorithms. Once tested in the development environment, every workflow goes through a production review where basic integration tests are performed and any necessary tuning is done. Post-review, the workflow is deployed onto the production Azkaban instance. The datasets and the tool suites are kept in sync across environments to allow easy reproduction of results.

6. EGRESS

The result of these workflows are usually pushed to other systems, either back for online serving or as a derived data-set for further consumption. For this, the workflows appends an extra job at the end of their pipeline for data delivery out of Hadoop.

This job consists of a simple 1-line command for data deployment. There are three main mechanisms available depending on the needs of the application:

- Key-value: the derived data can be accessed as an associative array or collection of associative arrays;
- Streams: data is published as a change-log of data tuples;
- OLAP: data is transformed offline into multi-dimensional cubes for later online analytical queries.

An interesting point is that the interface from the live service is agnostic to this data being generated offline. In other words, a feature can have its backend easily replaced with a real-time data consuming implementation with no change to the live service.

6.1 Key-Value

Key-value is the most frequently used vehicle of transport from Hadoop at LinkedIn and is made possible by the use of Voldemort. Voldemort is a distributed key-value store akin to Amazon's Dynamo [7] with a simple `get(key)` and `put(key, value)` interface. Tuples are grouped together into logical *stores*, which correspond to database tables. For example, a store for group recommendations will have a key being the member id of the recommendee and the value being the list of group ids that we recommend they join. Each key is replicated to multiple nodes depending on the preconfigured *replication factor* of its corresponding store. Every node is further split into logical partitions, with every key in a store mapped using consistent hashing to multiple partitions across nodes.

Voldemort was initially built for read-write traffic patterns with its core single node storage engine using either MySQL or Berkeley DB (BDB). With the introduction of Hadoop and the generation of derived data on HDFS, it became important to quickly bulk load data into the corresponding serving stores. The first use cases at LinkedIn were recommendation features where we store a list of recommendation entity ids and scored tuples as its values. Due to the dynamic nature of our social graph, the immutability of these values, and large changes in the score component between runs, it became important to completely replace the full key-space that was being served in Voldemort. Our initial prototype would build MyISAM, one of the storage engines of MySQL [25], database files on a per-partition basis as the last MapReduce job in a workflow, and then copy the data over into HDFS. Voldemort would then pull these files into a separate store directory and change the "table view" serving the requests to the new directory. This was not particularly scalable due to the data copy costs from the local task filesystem to HDFS and the overhead of maintaining and swapping views on MySQL tables.

Due to the pluggable nature of Voldemort, we were able to introduce a custom storage engine tailored to HDFS to solve the copy problem. The storage engine is made up of "chunk sets"—multiple pairs of index and data files. The index file is a compact structure containing a hash of the key followed by the offset to the corresponding value in the data file, with the entire file sorted by the hashed keys. A key lookup is achieved by running binary search within the index, followed by a seek into data file. A MapReduce job takes as its input a dataset and finally emits these chunk sets into individual Voldemort node-specific directories.

The identity mappers in this MapReduce job only emit the hash of the key "replication factor" number of times, followed by a custom partitioner that routes the key to the appropriate reducer responsible for the chunk set. The reducers finally receive the data sorted by the hash of the key and perform an append to the corresponding chunk set. This job supports producing multiple chunk sets per partition, thereby allowing one to tweak the number of reducers to increase parallelism. The simple sorted format of chunk sets permits fast construction.

On the Voldemort side, configurable number of versioned directories are maintained on a per-store basis with just one version serving live requests while others acting as backups. After generating new chunk sets on Hadoop, Voldemort nodes pull their corresponding chunk sets in parallel into new versioned directories. By adopting a pull methodology instead of push, Voldemort can throttle the data being fetched. A check is also performed with pre-generated checksums to verify integrity of pulled data. After the pull operation has succeeded, the chunk set files of the current live directory are closed and the indexes in the new chunk sets are memory mapped, relying on the operating system's page cache. This "swap" operation runs in parallel across machines and takes a sub-millisecond amount of time. The last step, after the swap, is an asynchronous cleanup of older versions to maintain the number of backups. Maintaining multiple backup versions of the data per store aids in quick rollback to a good state in case of either data or underlying algorithm problems.

This complete chunk generation, pull, and swap operation is abstracted into a single line Pig *StoreFunc* that is added by the user to the last job of their workflow. An example of this is shown below. This builds chunk sets from data available in "/data/recommendation", stores the node level output in a temporary location and then informs Voldemort to pull data from this directory. Finally, once the data is pulled completely by all the Voldemort nodes, a swap operation is performed.

```
recs = LOAD '/data/recommendations' USING AvroStorage();
STORE recs INTO 'voldemort://url' USING
 KeyValue('store=recommendations;key=member_id');
```

As chunk sets generated in Hadoop are immutable and are at the granularity of partitions, rebalancing of a Voldemort cluster is easy. The addition of new nodes is equivalent to changing ownership of partitions, which maps to moving corresponding chunk sets to the new node. This cheap rebalancing allows us to grow our clusters with increasing stores and request traffic. This is important, as it alleviates the need for stringent capacity planning: users push and change their stores and, as needed, the operations team grows and rebalances these clusters.

At LinkedIn, we maintain production and development read-only clusters. Researchers can deploy their data to the development cluster for testing purposes and as a way to hook data validation tools. We also run a "viewer application" that wraps common LinkedIn specific entities returned by the development cluster along with any researcher-specified metadata for displaying purposes. This provides an "explain" command around any derived data. For example, in a predictive analytics application, viewers usually show the feature weights before model application. This goes a long way into debugging issues, which can be difficult in a large and complex workflow, by providing data provenance.

Key-value access is the most common form of egress from the Hadoop system at LinkedIn. We have been successfully running these clusters for the past 3 years, with over 200 stores in production. We have seen consistent 99th percentile latency on most stores to be below sub 10ms.

A detailed description of the Voldemort key-value egress mechanism is described in Sumbaly et al. [31].

6.2 Streams

The second outlet for derived data generated in Hadoop is as a stream back into Kafka. This is extremely useful for applications that need a change log of the underlying data (e.g., to build their own data structures and indexes.)

This ability to publish data to Kafka is implemented as a Hadoop *OutputFormat*. Here, each MapReduce slot acts as a Kafka producer that emits messages, throttling as necessary to avoid overwhelming the Kafka brokers. The MapReduce driver verifies the schema to ensure backwards compatibility as in the ingress case (Section 4.1). As Kafka is a pull-based queue, the consuming application can read messages at its own pace.

Similar to Hadoop and the Voldemort case, a Pig abstraction is provided as a *StoreFunc* for ease of use. An example script to push a stream of session-based page-views (into the "SessionizedPageViewEvent" topic) is shown below. A registered schema is required.

```
sessions = FOREACH pageviews GENERATE Sessionize(*);
STORE sessions INTO 'kafka://kafka-url' USING
 Streams('topic=SessionizedPageViewEvent');
```

Due to Kafka's design for high throughput, there are no write-once semantics: any client of the data must handle messages in an idempotent manner. That is, because of node failures and Hadoop's failure recovery, it is possible that the same message is published multiple times in the same push. This is not a burden for most applications because they use the data in an idempotent manner as well (e.g., to update an index.)

6.3 OLAP

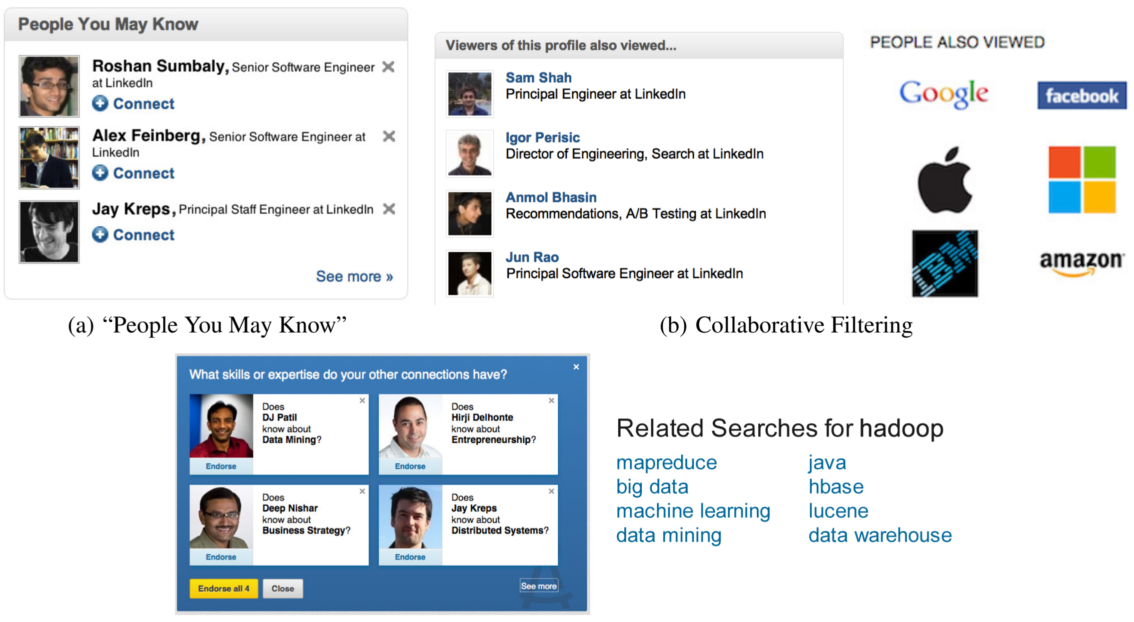
The final outlet offered is for structured multidimensional data (OLAP) for online queries. These online queries allow the end user to look at the data by slicing and dicing across various dimensions. Most enterprise OLAP solutions solve this problem by coupling the two required subsystems: cube generation and dynamic query serving. At LinkedIn, we have developed a system called Avatara [35] that solves this problem by moving the cube generation to a high throughput offline system and the query serving to a low latency system. By separating the two systems, we lose some freshness of data, but are able to scale them independently. This independence also prevents the query layer from the performance impact that will occur due to concurrent cube computation.

Given that the resulting data cubes are to be served in the request/response loop of a website, queries must be satisfied in the order of tens of milliseconds. Importantly, the use cases at LinkedIn have a natural shard key thereby allowing the final large cube to be partitioned into multiple "small cubes". As an example, to provide analytics on a member's activity, the cube could be sharded by member id as the product would not allow viewing results for another member.

To generate cubes, Avatara leverages Hadoop as the offline engine for running user-defined joins and pre-aggregation steps on the data. To ease the development, we provide a simple configuration file driven API that automatically generates corresponding Azkaban job pipelines. The underlying format for these small cubes are multidimensional arrays (i.e., arrays of tuples), where a tuple is combination of dimension and measure pairs. This compact format also makes it easy for the online engine to fetch the full data required in a single disk fetch. The output cubes are then bulk loaded into Voldemort as a read-only store for fast online serving. Due to its extensible architecture, Avatara is not tied to Voldemort and can support other online serving systems.

From the client query side, a SQL-like interface is provided to run computations like filtering, sorting, and aggregation operations, which the query engine optimizes by pushing down the predicate to the storage system. Because Avatara provides the ability to materialize at both the offline and online engine, the application developer has the option to choose between faster response time or flexible query combinations.

More details on Avatara are available in Wu et al. [35].


People You May Know

Roshan Sumbaly, Senior Software Engineer at LinkedIn
Connect

Alex Feinberg, Senior Software Engineer at LinkedIn
Connect

Jay Kreps, Principal Staff Engineer at LinkedIn
Connect

See more »

(a) "People You May Know"

Viewers of this profile also viewed...

Sam Shah
Principal Engineer at LinkedIn

Igor Perisic
Director of Engineering, Search at LinkedIn

Anmol Bhasin
Recommendations, A/B Testing at LinkedIn

Jun Rao
Principal Software Engineer at LinkedIn

PEOPLE ALSO VIEWED

(b) Collaborative Filtering

What skills or expertise do your other connections have?

Does DJ Patil know about Data Mining?
Endorse

Does Hirji Delhonte know about Entrepreneurship?
Endorse

Does Deep Nishar know about Business Strategy?
Endorse

Does Jay Kreps know about Distributed Systems?
Endorse

Endorse all 4 | Close | See more

(c) Skill Endorsements

Related Searches for hadoop

mapreduce	java
big data	hbase
machine learning	lucene
data mining	data warehouse

(d) Related Searches

Figure 2: Examples of features using the Voldemort key-value egress mechanism. In these examples, the key is usually some type of identifier (e.g., a member or company) with the value being a set of recommendations.

7. APPLICATIONS

Most features at LinkedIn rely on this data pipeline either explicitly—where the data is the product, or implicitly—where derived data is infused into the application. In this section, we cover several of these features. All of these applications leverage the Kafka ingress flow and use Azkaban as their workflow and dependency manager to schedule their computation at some schedule (e.g., every 4 hours) or based on some trigger (e.g., when new data arrives).

7.1 Key-Value

Key-value access using Voldemort is the most common egress mechanism from Hadoop. Over 40 different products use this mechanism and it accounts for approximately 70% of all Hadoop data deployments at LinkedIn.

People You May Know.

"People You May Know," as shown in Figure 2a, attempts to find other members a user may know on the social network. This is a link prediction problem [18] where one uses node and edge features in the social graph to predict whether an invitation will occur between two unconnected nodes.

The Hadoop workflow has evolved into a large set of feature extraction tasks—signals such as the number of common connections, company and school overlap, geographical distance, similar ages, and many others—followed by a model application step. As of this writing, there are close to a 100 Azkaban tasks and therefore several hundred MapReduce jobs to build these recommendations every day. Azkaban helps manage the size and complexity of this workflow: the engineers on this project divide into particular sub-workflows where they are each most knowledgeable.

The execution of this workflow is also managed by Azkaban and alerts are sent to LinkedIn's operations team if anything goes awry. The team conducts numerous split or A/B tests and can stitch in additional signals into the model, test the workflow on the development Hadoop grid, then quickly publish the new workflow into the executing system if the results are satisfactory.

The resulting data model of this workflow is to push a key-value store where the key is a member identifier and the value is a list of member id, score pairs. The online service for this feature retrieves these results from the Voldemort store through a simple `get`, applies business rules, before the frontend service decorates the results and presents them to the user.

Collaborative Filtering.

Co-occurrence or association rule mining [1] results are shown on the website as a navigational aid for the member to discover related or serendipitous content from the "wisdom of the crowd." For example, on a member's profile, co-occurrence of other viewed profiles are displayed as shown in Figure 2b. Similarly, on a company entity page, co-occurrence of other companies users could view are displayed.

This pipeline initially computed only member-to-member co-occurrence, but quickly grew to meet the needs of other entity types, including cross-type (e.g., member-to-company) recommendations. LinkedIn's frontend framework emits activity events on every page visit as part of LinkedIn's base member activity tracking. A parameterized pipeline for each entity type uses these events to construct a co-occurrence matrix with some entity specific tuning. This matrix is partially updated periodically depending on the needs of each entity type (e.g., jobs are ephemeral and refresh more frequently than companies, which are relatively static.) The resulting key-value store is a mapping from an entity pair—the type of the entity and its identifier—to a list of the top related entity pairs.

Skill Endorsements.

Endorsements are a light-weight mechanism where a member can affirm another member in their network for a skill, which then shows up on the endorsed member's profile. As part of this feature, recommendations on who to endorse are available as shown in Figure 2c.

A workflow first determines skills that exist across the member base. This is a deep information extraction problem, requiring deduplication, synonym detection (e.g., “Rails” is the same as “Ruby on Rails”), and disambiguation (e.g., “search” could mean “information retrieval” in a computer science sense or “search and seizure” in a law enforcement sense.) From this taxonomy, another joins profile, social graph, group, and other activity data to determine the canonicalized skills for a member. These stores are pushed as a key-value store wrapped by a service that exposes an API for any client to determine a member’s skills and resolve skill identifiers. Once skills are resolved, another workflow computes the endorsement recommendations through a model that combines the propensity for a member to have a skill and the affinity between two members. The resulting recommendations are delivered as a key-value store mapping a member id to a list of member, skill id, and score triples. This data is used by the frontend team to build the user experience.

Related Searches.
LinkedIn’s related search system [27] builds on a number of signals and filters that capture several dimensions of relatedness across member search activity. Related search recommendations, as shown in Figure 2d for the query “Hadoop”, provide an important navigational aid for improving members’ search experience in finding relevant results to their queries. The search backend emits search activity events and the resulting store is keyed by search term and, because the website is international, the locale.

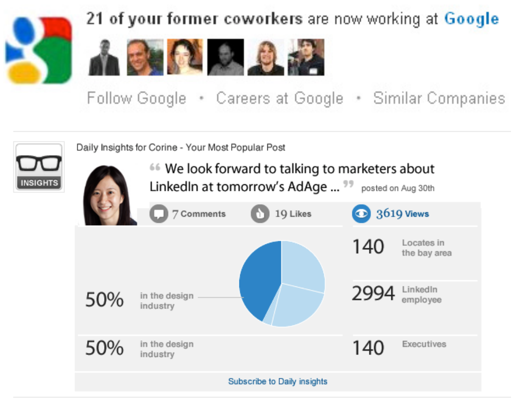
7.2 Streams

Applications that require a push-based model leverage Kafka as the delivery mechanism. Both online and offline systems negotiate a predefined Kafka topic, with the offline system producing and the online system consuming the output. The stream egress mechanism represents around 20% of all Hadoop-based data deployments at LinkedIn.

News Feed Updates.
On most consumer websites, news feed generation is driven by an online system. Updates are generated on a per-member basis based on interactions with other components in the ecosystem. For example, a LinkedIn member receives an update when one of their connection updates their profile. To show deeper analytical updates requires joining data across multiple sources, which can be time consuming in a service-oriented manner. For example, to generate an update highlighting the company that most of a member’s former coworkers now work for (as shown Figure 3a), requires joining company data of various profiles. As this is a batch compute-intensive process, it is well suited for an offline system like Hadoop. Further, ease of data processing in the Hadoop ecosystem engenders quick prototyping and testing of new updates.

Each of these offline updates are maintained by workflows running at different scheduled intervals. The output of each workflow writes its data in a packaged format to a news feed Kafka topic using the functionality mentioned in Section 6.2. The news feed system listens to this topic and stores the updates for online serving, blending them with online-generated updates.

Email.
Similar to the news feed system, email generation can be either online or offline. Certain types of emails (e.g., password recovery or joining a LinkedIn group), are activity driven and require online generation and delivery. On the other hand, a digest email is well suited for an offline system. Besides ease of development, running the computation online results in a bursts of calls to backend services, impact site performance.


21 of your former coworkers are now working at Google

Follow Google • Careers at Google • Similar Companies

Daily Insights for Corine - Your Most Popular Post

“We look forward to talking to marketers about LinkedIn at tomorrow’s AdAge ...” posted on Aug 30th

7 Comments 19 Likes 3619 Views

50% in the design industry

50% in the design industry

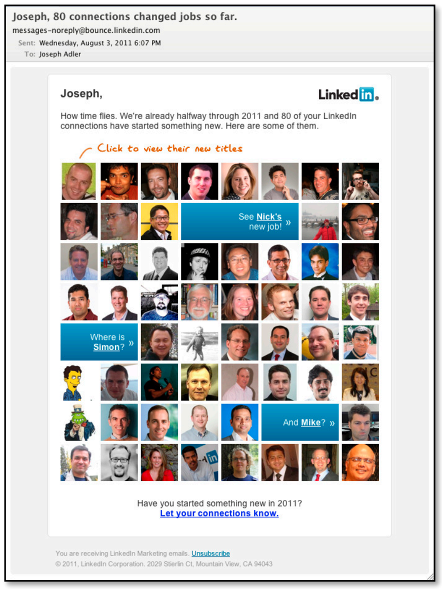
140 Locates in the bay area

2994 LinkedIn employee

140 Executives

Subscribe to Daily insights

(a) News Feed Updates


Joseph, 80 connections changed jobs so far.
messages-noreply@bounce.linkedin.com
Sent: Wednesday, August 3, 2011 6:07 PM
To: Joseph Adler

Joseph,

How time flies. We're already halfway through 2011 and 80 of your LinkedIn connections have started something new. Here are some of them.

Click to view their new titles

See Nick's new job! »

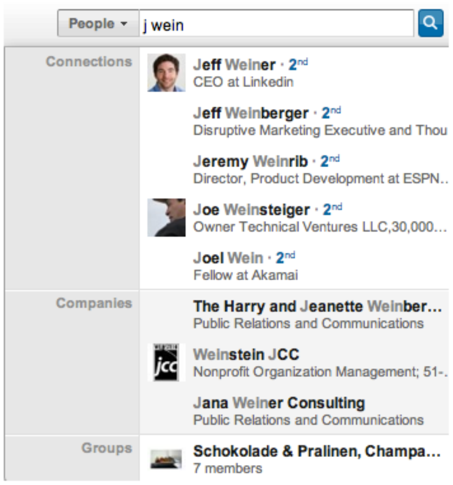
Where is Simon? »

And Mike? »

Have you started something new in 2011?
Let your connections know.

You are receiving LinkedIn Marketing emails. Unsubscribe
© 2011, LinkedIn Corporation. 2029 Stierlin Ct, Mountain View, CA 94043

(b) Email


People ▾ j wein

Connections
Jeff Weiner · 2nd
CEO at Linkedin
Jeff Weinberger · 2nd
Disruptive Marketing Executive and Thou
Jeremy Weinrib · 2nd
Director, Product Development at ESPN..
Joe Weinsteiger · 2nd
Owner Technical Ventures LLC,30,000...
Joel Wein · 2nd
Fellow at Akamai

Companies
The Harry and Jeanette Weinber...
Public Relations and Communications
Weinstein JCC
Nonprofit Organization Management; 51-.
Jana Weiner Consulting
Public Relations and Communications

Groups
Schokolade & Pralinen, Champa...
7 members

(c) Typeahead

Figure 3: Examples of features where derived data is streamed out of Hadoop into another system using Kafka.

For example, the email shown in Figure 3b shows a member's connections that have started a new job in the past 6 months. This is a fairly straightforward Pig script taking less than 100 lines of code: one must take a member's connections, join the requisite profile data, and run a filter to determine new positions.

The architecture of the email system is similar to the network update stream with a workflow populating a Kafka stream with the necessary email content and envelope information. The email system packages this data with the appropriate template into an email message that is eventually sent to the user.

Relationship Strength.

The edges in LinkedIn's social graph are scored by a periodic offline job as a means to distinguish between strong and weak ties [10]. These scores are periodically published to a stream and are read by numerous consumers to populate their own indexes.

For example, LinkedIn's social graph service, which supports a limited set of online graph traversals (e.g., degree distance or shortest paths between nodes), indexes these scores to support graph operations such as finding the "best" path in the graph. As another example, LinkedIn supports a search typeahead [17] that attempts to auto-complete partial queries, as shown in Figure 3c. As another consumer of this relationship strength stream, the typeahead service indexes and uses these scores to display entities that are more relevant by strength to the member.

7.3 OLAP

The structured nature of LinkedIn profiles provides various member facets and dimensions: company, school, group, geography, etc. Combined with activity data, this can provide valuable insights if one can slice and dice along various dimensions. OLAP accounts for approximately 10% of egress usage.

Who's Viewed My Profile?.

LinkedIn provides users with the ability to view statistics around members who viewed their profile. For example, in Figure 4a, profile views are shown faceted by time, industry, and country. Because this dashboard is generated on a per-member basis, small cubes are computed offline keyed by member id.

The data generation workflow starts by aggregating profile views to a coarser "week" level granularity, followed by a join with various individual dimension datasets. The granularity of the offline rollup dictates the flexibility of queries online; in this case, any rollups must be at a minimum of a week level.

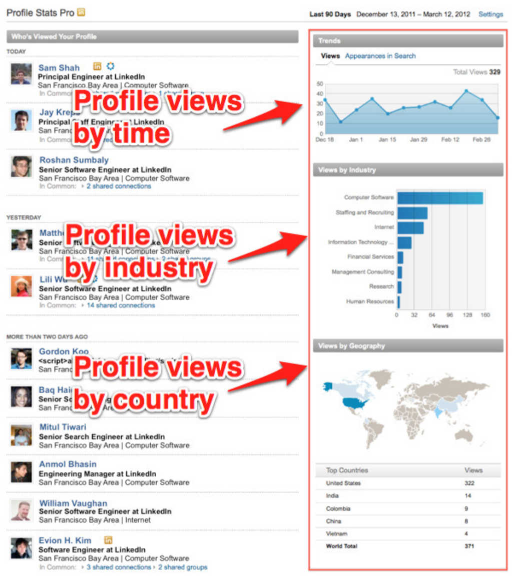
The output—a small cube—is a partially materialized list of tuples of the form (viewer_id, industry_id, country_id, timestamp, count_measure), where the first 3 fields signify the dimensions followed by the measure. After this is loaded into a Voldemort read-only store, the online query engine allows further real-time group-by, order, and limit requests to retrieve the top dimensions. This in-memory manipulation of data is very quick, with 90% of queries being satisfied in under 20 ms.

Who's Viewed This Job?.

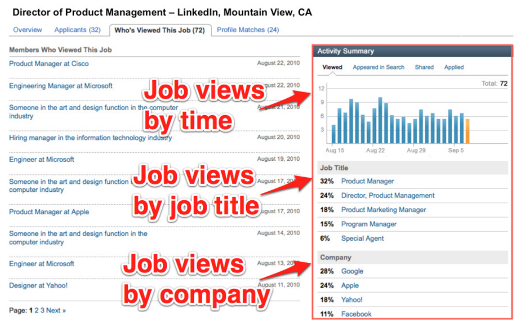
Figure 4b shows an analytical dashboard of members who viewed a recruiter's job, delineated by title, time, and company. The job id is the primary key of the cube, with a similar workflow as "Who's Viewed My Profile?," to generate a cube for 3 dimensions on job views.

8. CONCLUSION AND FUTURE WORK

In this paper, we presented the end-to-end Hadoop-based analytics stack at LinkedIn. We discussed how data flows into the offline system, how workflows are constructed and executed, and the mechanisms available for sending data back to the online system.



(a) "Who's Viewed My Profile?"



(b) "Who's Viewed This Job?"

Figure 4: Examples of features where different flavors of descriptive statistics are exposed through multidimensional queries using the OLAP egress mechanism.

As the processing and storage cost of data continues to drop, the sophistication and value of these insights will only accumulate. In developing this system, we strived to enable non-systems programmers to derive insights and productionize their work. This empowers machine learning researchers and data scientists to focus on extracting these insights, not on infrastructure and data delivery.

There are a several avenues of future work. MapReduce is not suited for processing large graphs as it can lead to sub-optimal performance and usability issues. Particularly, MapReduce must materialize intermediate data between iterations and the stateless map and reduce nature yields clunky programming as the graph must be passed between stages and jobs. This is an active area of research that we are investing in.

Second, the batch-oriented nature of Hadoop is limiting to some derived data applications. For instance, LinkedIn provides news article recommendations and due to the ephemeral nature of news (we need to support "breaking news"), it is built on a separate customized platform. We are developing a streaming platform that will hook into the rest of the ecosystem to provide low-latency data processing.

Acknowledgements

The authors are indebted to the numerous engineers from the LinkedIn data team that have contributed to the work presented in this paper, our grid operations team for their exemplary management and guidance, and the invaluable feedback from our data scientists.

References

[1] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216, 1993.

[2] A. Barbuzzi, P. Michiardi, E. Biersack, and G. Boggia. Parallel Bulk Insertion for Large-scale Analytics Applications. In *Proceedings of the 4th International Workshop on Large Scale Distributed Systems and Middleware (LADIS)*, pages 27–31, 2010.

[3] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, and A. Aiyer. Apache Hadoop goes realtime at Facebook. In *SIGMOD*, pages 1071–1080, 2011.

[4] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A Distributed Storage System for Structured Data. In *OSDI*, 2006.

[5] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, and D. Sampath. The YouTube video recommendation system. In *RecSys*, pages 293–296, 2010.

[6] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *OSDI*, 2004.

[7] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's Highly Available Key-Value Store. *SIGOPS Operating Systems Review*, 41: 205–220, Oct. 2007.

[8] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[9] L. George. *HBase: The Definitive Guide*. O'Reilly Media, 2011.

[10] E. Gilbert and K. Karahalios. Predicting tie strength with social media. In *CHI*, pages 211–220, 2009.

[11] K. Goodhope, J. Koshy, J. Kreps, N. Narkhede, R. Park, J. Rao, and V. Y. Ye. Building LinkedIn's Real-time Activity Data Pipeline. *IEEE Data Eng. Bull.*, 35(2):33–45, 2012.

[12] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. ZooKeeper: wait-free coordination for Internet-scale systems. In *USENIX*, 2010.

[13] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer. Enterprise Data Analysis and Visualization: An Interview Study. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2917–2926, 2012.

[14] I. Konstantinou, E. Angelou, D. Tsoumakos, and N. Koziris. Distributed Indexing of Web Scale Datasets for the Cloud. In *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud (MDA)*, pages 1:1–1:6, 2010.

[15] J. Kreps, N. Narkhede, and J. Rao. Kafka: A Distributed Messaging System for Log Processing. In *Proceedings of the NetDB*, 2011.

[16] G. Lee, J. Lin, C. Liu, A. Lorek, and D. Ryaboy. The unified logging infrastructure for data analytics at Twitter. *Proc. VLDB Endow.*, 5(12):1771–1780, Aug. 2012.

[17] G. Li, S. Ji, C. Li, and J. Feng. Efficient type-ahead search on relational data: a TASTIER approach. In *SIGMOD*, pages 695–706, 2009.

[18] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *CIKM*, pages 556–559, 2003.

[19] J. Lin and A. Kolcz. Large-scale machine learning at Twitter. In *SIGMOD*, pages 793–804, 2012.

[20] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, Jan. 2003.

[21] P. Mika. Distributed Indexing for Semantic Search. In *Proceedings of the 3rd International Semantic Search Workshop (SEMSEARCH)*, pages 3:1–3:4, 2010.

[22] A. Nandi, C. Yu, P. Bohannon, and R. Ramakrishnan. Distributed Cube Materialization on Holistic Measures. In *ICDE*, pages 183–194, 2011.

[23] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: a not-so-foreign language for data processing. In *SIGMOD*, pages 1099–1110, 2008.

[24] S. Owen, R. Anil, T. Dunning, and E. Friedman. *Mahout in Action*. Manning Publications Co., Greenwich, CT, USA, 2011.

[25] S. Pachev. *Understanding MySQL Internals*. O'Reilly Media, 2007.

[26] A. Rabkin and R. Katz. Chukwa: a system for reliable large-scale log collection. In *LISA*, pages 1–15, 2010.

[27] A. Reda, Y. Park, M. Tiwari, C. Posse, and S. Shah. Metaphor: a system for related search recommendations. In *CIKM*, 2012.

[28] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, Washington, DC, USA, 2010.

[29] A. Silberstein, B. Cooper, U. Srivastava, E. Vee, R. Yerneni, and R. Ramakrishnan. Efficient Bulk Insertion into a Distributed Ordered Table. In *SIGMOD*, pages 765–778, 2008.

[30] A. Silberstein, R. Sears, W. Zhou, and B. Cooper. A batch of PNUTS: experiences connecting cloud batch and serving systems. In *SIGMOD*, pages 1101–1112, 2011.

[31] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah. Serving Large-scale Batch Computed Data with Project Voldemort. In *FAST*, 2012.

[32] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, and R. Murthy. Hive—a petabyte scale data warehouse using Hadoop. In *ICDE*, pages 996–1005, 2010.

[33] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. S. Sarma, R. Murthy, and H. Liu. Data warehousing and analytics infrastructure at Facebook. In *SIGMOD*, pages 1013–1020, 2010.

[34] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, 2010.

[35] L. Wu, R. Sumbaly, C. Riccomini, G. Koo, H. J. Kim, J. Kreps, and S. Shah. Avatara: OLAP for web-scale analytics products. *Proc. VLDB Endow.*, pages 1874–1877, 2012.