Appeared in *Proceedings of the 14th Workshop on Hot Topics in Operating Systems (HotOS '13)*
Santa Ana Pueblo, New Mexico, May 2013

Making Every Bit Count in Wide-Area Analytics

Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek Pai, and Michael J. Freedman
Princeton University

Abstract

Many data sets, such as system logs, are generated from widely distributed locations. Current distributed systems often discard this data because they lack the ability to backhaul it efficiently, or to do anything meaningful with it at the distributed sites. This leads to lost functionality, efficiency, and business opportunities. The problem with traditional backhaul approaches is that they are slow and costly, and require analysts to define the data they are interested in up-front. We propose a new architecture that stores data at the edge (i.e., near where it is generated) and supports rich real-time and historical queries on this data, while adjusting data quality to cope with the vagaries of wide-area bandwidth. In essence, this design transforms a distributed data collection system into a distributed data analysis system, where decisions about collection do not preclude decisions about analysis.

1 Introduction

Recent years have seen an explosion in the number and variety of devices producing data streams, from software logs to handheld phones to aerial cameras. About 2.5 quintillion bytes of data are created globally each day [1]. Much of this data starts its life widely distributed. However, users often want to analyze data across the system as a whole. By focusing only on data processing within datacenters, the research community is overlooking an increasingly important part of the Big Data challenge.

Today, a common technique for data analysis is to backhaul all the data generated at wide-area sources to a central datacenter, where it is then stored and processed. This approach allows analysts to use existing tools developed for single-datacenter large-scale analytics. Backhaul, however, incurs the high cost associated with wide-area data transfer. This tradeoff is a bad one given historical and contemporary trends in computing cost.

Historically, local storage and processing costs have been dropping faster than WAN costs. For example, prices from 2003-2008 of wide-area bandwidth to large ISPs dropped only 2.7x, while CPU and storage dropped 16x and 10x, respectively [5].

A few short-term factors are slowing the decline in some of these areas, but should have little impact on the long-term cost trend: disk prices have been impacted by supply chain problems caused by floods in Thailand [20], and bandwidth prices are being temporarily suppressed by recession economics and government subsidies [22]. So in the short term, WAN bandwidth prices are expected to decline at about 25% per year [26], while CPU and disk prices should fall 15-30% and 10-20% per year [20].

In the long run, though, the bottleneck for wide-area bandwidth is capacity at trans-oceanic crossings, and the fundamentals there have not changed: from 2007 to 2011, transatlantic cable bandwidth grew at a 19% annual rate [25], and prices only fell insignificantly. The high capital costs for laying trans-oceanic cables are unlikely to change significantly; the bandwidth available through these cables is also unlikely to increase quickly.

These cost trends point to the fact that it is simply not economical to copy all the data that is collected globally to a central location. The question that analysts face is how to prioritize the data that is transferred across the wide-area for analysis. A common approach is to create a static policy of which data to backhaul, but this approach forces analysts to commit, in advance, to a valuation of the data. The value of data is hard to predict, however, and may also change over time [14]. For example, monitoring data may either be vital for debugging, or pointless trivia, depending on the context. While some wide-area stream-processing systems (e.g., Hourglass [21]) reduce data volumes by aggregating near the source, they still suffer from the same flaw of requiring users to decide up-front which data to transfer.

As a motivating example, suppose a content distribution network operator wants to know the 100 most popular domains, each minute. A naive approach is for each node to send the popularity of each domain to a central location for analysis, each minute. This is wasteful of bandwidth. An optimized, though sometimes inaccurate, approach is to send only the top-k domains, for some $k > 100$. Now suppose the popularity of some domain surges. An analyst might wish to inspect the history of the domain, before it entered the top 100. However, that data is unavailable because it was never backhauled.

A great deal of valuable data is never collected due to costs. Many CDNs discard or do not collect low-level connection monitoring and progress information, for example. Such data has value, but backhauling it incurs high eco-

nomic costs and is slow when system resources are under load and bandwidth is limited, precisely the times when monitoring data is needed typically most. (We use CDNs as a running example due to our familiarity with them. Additional widely distributed data sources are discussed in the next section.)

Ignoring the changing value of data necessarily leads to mischosen data, which translates to lost functionality, efficiency, and business opportunities. A better analysis system would let user queries transparently span data at the edge and data that has been backhauled.

This paper proposes a new architecture for wide-area data analysis. Rather than statically defining the data to be collected and backhauled, we envision systems that prioritize data in the presence of changing requirements and resources, and that can re-plan if the perceived value of data changes. Realizing this vision requires wide-area analysis systems to embrace three high-level design choices:

Store data near the edge, either within the local network where the data is generated, or as close to it as physically possible, such as points of presence or compute platforms co-located with sensors. This retains data in case its perceived value changes and it becomes subsequently worth backhauling for further processing.

Incorporate aggregation and approximation as first-class primitives to reduce data volume while preserving value. The user needs to express how data will be used and the requirements on data quality. This extra information can allow the system to degrade gracefully when resources are limited, giving approximate answers promptly and refining them later if desired and possible.

Use structured storage that tracks data quality. We envision using data cubes from online analytical processing (OLAP) as a unifying abstraction for tracking incrementally-refined data.

A system that incorporates these techniques can cope with expensive and unreliable wide-area bandwidth. It will move data only when necessary, prioritize data based on its value, and compensate when bandwidth becomes more available.

2 Uses

We now provide some example applications that could benefit from efficient analysis of large, dynamic, wide-area data sets to illustrate the importance of the topic.

Content analytics: CDNs can generate very large volumes of log data (e.g., recording full HTTP header data for each request), spread across widely distributed servers. Yet detecting hot spots or popular items requires only aggregate statistics, while forensic analysis of anomalies typically involves only a few servers in detail.

Debugging logs: Systems can keep extensive logs of their internal behavior to enable debugging. This data is useful only when an issue is detected; centralizing all debugging data may be overly costly in the normal case. On the other hand, centralizing aggregates derived from such logs may also be useful, since statistical methods for log analysis have become increasingly effective [19].

Adaptive resource control: Many large distributed systems collect nodes' performance data to better manage and coordinate resources. Having all nodes report to a centralized controller does not scale well, especially in the wide area. Rather than building ad-hoc monitoring, a general-purpose data-stream analysis engine can efficiently track resource utilization across the entire system.

Imaging data: Highways and public places are increasingly festooned with cameras, mounted on both fixed and aerial platforms. While the data collected is large images or video, the answers to queries of interest are often much smaller (e.g., "what was the average speed of traffic on this stretch of highway over the last month?" or "where are there unexpectedly large gatherings of people?"). In many cases, there are no technical barriers (space, power, etc.) to installing large compute or storage resources near the point of data generation. For example, IP-based closed-circuit television (CCTV) cameras support recording to on-site network-attached storage, or to internal flash. In contrast, bandwidth to widely dispersed sites (sometimes over cellular or satellite links) is expensive and likely to remain so.

Medical records: The barrier to backhauling data is not always cost. There are many useful queries one might wish to run on medical records data, such as searching for population-wide trends or trying to assess the frequency of a given genetic variation. However, a centralized national medical records repository poses significant privacy, economic, and regulatory challenges (as does centralization in many other application domains). Past work has addressed data integration and schema changes in federated query systems, yet efficient standing queries remain an open problem [13].

The above applications all share a common thread: (i) high data volumes are generated across the wide-area and (ii) most analysis tasks require a small fraction of this data or generate summaries that are much smaller than the raw data. In each application, there will be some computation and storage resources at each site where data is generated, and some centrally.

We envision two classes of queries: (i) **ad-hoc queries** that use stored data to produce detailed information about past events, and (ii) **standing queries** that continuously update their results as new data comes in. In practice, the two are not only both necessary, but are mutually dependent. One-off queries execute more quickly and reliably if they can use continuously-maintained centralized aggregates, rather than touching many edge storage locations. Standing queries can indicate surprises or anomalies that are investigated with ad-hoc queries.

3 Requirements of wide-area analytics

To understand the requirements that shape design decisions for a wide-area analytics system, we discuss some key factors and how they differ from local-area systems.

Local domains as visible abstractions. We expect users will have strong views about which data will be stored in what locations, and when data will be backhauled. Like the decision of which database indices to maintain, the decision of what to backhaul will shape the performance landscape of subsequent queries. As a result, we foresee the need for novel "region" abstractions that group related systems and allow the user to explicitly define which data is transferred between them. Regions can even represent a highly-connected local area domain, such as a cluster or a point-of-presence, or may define legal domains with specific data retention and reporting policies. Analysis systems cannot conceal region boundaries since users need to control what data is stored where.

In comparison, systems like MapReduce expose the boundaries between both task instances and separate clusters, but do not explicitly expose nodes or racks to users. Related systems may internally use rack- or node-locality to optimize performance, but users do not refer to these boundaries when defining computations to perform.

Dealing with bandwidth variability. Local-area analytics clusters can be provisioned to match the worst-case expected inputs, since compute and local networking resources are predictable and reliable. In contrast, wide-area bandwidth availability may be affected by shared lines and congestion, while the bandwidth demands of a service may be affected by diurnal variation, unexpected peak demands on the service, or popularity differences among the nodes of a geographically-distributed service. When available network resources become too scarce to transfer all the data for a standing query, system designers are faced with a choice: that query must either abort, fall ever farther behind, or be modified to stay within the resource limits. We believe the right approach is to alter the query, delivering as much data quality as possible to the user.

We refer to these query modifications as adaptation, and think it is likely to be a characteristic of wide-area streaming analytics. In the local domain, processing systems can mitigate stragglers via speculative execution. This strategy does not help if wide-area bandwidth is the bottleneck. Since data is not replicated across the wide area, there is no alternate location where a task can be usefully relocated; stragglers and data unavailability have to be mitigated in some other way. Possible adaptation strategies may include keeping only a sample of incoming data, sending data less often (e.g., emitting aggregate statistics every minute instead of every second), or even filtering data, discarding records judged likely irrelevant.

Network resources can fluctuate on a minute-by-minute and hour-by-hour basis, and applications like real-time anomaly detection and load balancing require up-to-date information on timescales of seconds. As a result, an analysis system for these purposes must continuously probe bandwidth and quickly detect if it is falling behind. In contrast, local-area bandwidth is relatively predictable, and systems can use admission control to prevent overloads.

The sort of dynamic tracking and query adaptation we envision cannot be easily deployed atop current frameworks. MapReduce, streaming databases, and most other distributed systems do not collect or expose resource usage at fine granularities and do not have mechanisms for modifying currently-running jobs.

Backfill to improve data. Some current streaming systems support backfill, i.e., situations in which a query result has already been output, but then the result changes due to the arrival of new data. In wide-area systems, however, not only do we expect backfill to be more frequent due to the variability of wide-area systems, but we also expect greater delays between initial results and the arrival of more data. As a result, it makes sense to think of backfill as a first-class part of the system, which can be intentionally exploited to make the system more adaptive.

While backfill naturally arises from things like bandwidth shortages and temporary failures, it can also be paired with approximation. When a system encounters a bandwidth shortage, it can send approximate or coarse data immediately, and then fill in precise values later if and when bandwidth allows. This approach allows time-shifting bulk data transfer of raw records, while still allowing real-time analytics. Backfill ensures that the long-term quality of the data is not impaired.

As a result, backfill cannot simply be bolted onto existing processing frameworks; it must be considered early in the design process. Enforcing policy requires user specification, a global view of running queries, and the ability to change the data quality requirements for one query to support new downstream queries. Past research has looked at efficient incremental mechanisms for backfill [4, 9, 17, 18], and we expect to be able to leverage their design efficiencies.

4 Structuring Wide-Area Computation

We now discuss approaches for incorporating widely distributed storage into analytics systems, and processing steps that can make it feasible and bandwidth adaptive.

Structuring storage with cubes. We believe that analytics systems should incorporate structured storage in ways that simplify analysis tasks. Analysts working with online analytics processing (OLAP) databases often find it useful to represent data using a structure called an OLAP cube, which is a multi-dimensional array that encapsulates

numerical properties and relationships between fields in structured input data, similar to a database relation [10]. It is defined by a set of dimensions, which specify the coordinates of an array cell, and a set of aggregates, which specify the statistics stored in a cell. Each dimension indexes some properties of the input data, such as the URL or time period of web requests. The cells addressed by these dimensions contain statistics about a given URL and time period: for example, the total number of requests and the maximum request latency.

Unlike a relational database, the aggregates are part of schema. Each field in a cell is associated with a particular aggregation function, such as `count` or `max`. When new data is added is added to a cell (i.e., a new web request comes in), that aggregation function is applied to the old and new values, and the result stored back into the cell. Aggregation functions yield the same result regardless of the input data order.

A key benefit of the cube abstraction is that it decomposes gracefully: given two cubes with the same schema, there is one unambiguous way to merge them. Cells with the same dimension values get merged together by combining their partial aggregates. As a result, a cube-based system can reason about aggregation trees, and can introduce partial aggregation without altering the result. This is not possible in a purely key-value or relational model.

Using data cubes as a storage abstraction does not constrain the choice of programming model. Today's streaming systems often use a dataflow model, in which data flows through a network of operators that transform the data flowing past [2, 6, 8, 24]. One can integrate cubes into this model, treating them as part of the dataflow graph. One might also write purely declarative SQL-style programs for querying or transforming cubes. One might even combine cubes with a data-parallel imperative programming model.

Reasoning about data quality. Wide-area streaming analytics will sometimes have to trade away some data quality to reduce bandwidth consumption. Analytics systems, particularly if they are dynamically adjusting data quality, need to track the ways in which data has been degraded. This lets the system put error bounds on query results. It also lets the system notify the user if data is not available at the requested granularity but is available in a less precise form.

Different data sources may have different quality levels. In a CDN, for example, some nodes may be at peak diurnal usage and sending only rough data, while other nodes may be lightly loaded and sending complete data. The analytics system must be able to gracefully combine these disparate intermediate inputs.

Coarsening is an adaptation that fits particularly well with the cube model. In this context, coarsening means keeping aggregate statistics about larger items, such as per-minute instead of per-second data. The cube data model makes it easy to reason about and support coarsening, since a cube cell can explicitly indicate the range of dimension values that it covers. If the input data is at different granularities, the system can aggregate all the data together at the coarsest level of granularity.

Coarsening is not the only useful adaptation that reduces data precision. Data might also be sampled or filtered according to (potentially complex) criteria. To help the user assess a system's output, the system must keep enough metadata to explain the accuracy and lineage of data. The cube model, by making aggregation explicit, helps. In addition, addressing collections of cells through dimension value ranges is a compact way of tracking and querying metadata.

Incorporating explicit degradation policies. A wide-area system can dynamically adjust queries in several different ways, where the right strategy depends on the query and the user's needs. Analytics systems can ease the pain of articulating such policies, via interactive tools for authoring and reasoning about degradation policies.

Data quality has many aspects, and is not a simple linear scale [15]. A delay of ten seconds might be acceptable for one query but not another. Sampling is most effective for aggregates that are not sensitive to outliers. No single strategy is appropriate for all queries. Instead, users will need a way to specify the appropriate bandwidth-conservation strategy for each query.

Reasonable policies may be complex. A user might desire the following policy: "Under normal circumstances, report a histogram of latencies every second. If bandwidth is scarce, the reporting interval should be gradually increased up to every thirty seconds. Beyond that point, the reporting interval should remain unchanged, but the histogram itself should be coarsened (represented with fewer buckets)." A system's policy language should be expressive enough to represent policies of this sort.

The policy language should also be extensible. The scope of possible data degradations is very large and users will sometimes need to introduce new degradations. Consider the case of quantiles (such as medians). Quantiles are a so-called holistic aggregate that require storage overhead proportional to the size of the full data set [16]. To compensate, a wide variety of approximation (synopsis) techniques have been developed, appropriate for different data volumes, distributions, and accuracy goals [7]. Users should be able to add additional synopsis techniques tailored to their specific needs, and have the analytics system use these techniques appropriately.

Optimizing across queries. Analytics systems typically have many concurrent queries. Multi-query optimization has therefore been extensively researched [23]. The topic

takes on new aspects, however, in a system with widely distributed storage and with data at multiple fidelities. Today's cross-query optimizers look for sub-queries that can be reused. In wide-area analytics, the cost of reusing a sub-query depends strongly on where it is computed, and on the resultant precision.

Consider the following concrete example, to see how precision can enter into these calculations. Suppose that data from several sources is being copied to a central location and then combined. If one source is sending data at five-second granularity, while another is sending data every second, then the combined data will only be at five-second granularity. Depending on the situation, it might or might not be useful to keep a copy of the fine-grained disaggregated data. If future backfill is likely to replace the coarse data with higher-precision values, for example, then it would make sense to keep the central copy of the fine-grained data. The decision of which data to retain (and where) depends on whether a future computation will use the data.

5 Related Work

Our work is inspired by existing data stream management systems, such as Borealis and Storm [2, 6, 8, 24]. These systems do not handle storage, but instead aim to process incoming updates with minimal latency. In contrast, our interest is in processing dispersed and changing data sets, in the presence of network limitations and failures.

Most streaming systems were designed for single-datacenter deployments, but there has been some work on wide-area deployment [12, 21]. That work assumed that computation resources, sources, and destinations were scattered around ad-hoc, e.g., as PlanetLab nodes. As a result, sophisticated techniques were needed for placement. We believe this assumption is too pessimistic and that placement decisions can be handled relatively simply. For our applications, there will generally be only two or three options for data placement: the site where the data is generated, the nearest point-of-presence, or else a centralized datacenter. Placement within a datacenter or point-of-presence can be delegated to existing scheduling systems or scale-out processing engines. As a result, it will be sufficient for users to give guidance about data placement, and the system can then evaluate the (small) set of reasonable options for placing computation.

There has been a great deal of work on scalable processing systems for the datacenter. This includes traditional databases, MapReduce, and a range of systems in between. As we noted above, this work is predicated on flat, reliable, and high-performance networks. Much of this work is therefore inapplicable in the wide area, particularly if wide-area replication is to be avoided.

We are investigating system architectures that reduce data volumes while minimizing the reduction in accuracy. Similarly, BlinkDB [3] deploys sampling-based approximations on top of MapReduce and Hive to reduce latency. In BlinkDB, the data is carefully pre-sampled with specific statistical goals; small probing jobs are used to estimate query run-time. In contrast, continuous wide-area analytics systems will have to measure and adapt to available bandwidth, without the benefit of a prior data-import step. We also envision a range of degradation techniques, not just sampling.

Tree aggregation can be used to reduce bandwidth without reducing accuracy. The sensor network community has used this technique extensively for power-constrained devices, such as in the seminal Tiny Aggregation Service [22]. Much subsequent work looked at using redundant routes to compensate for unreliable connections and faulty nodes. In contrast, the applications we envision use hardware that is not power-constrained, and we expect conventional IP networking to deliver suitable routes.

There has been substantial recent work on incremental view maintenance [4, 9, 11, 17, 18], allowing databases to promptly incorporate updates. We expect many of the ideas from this work to be applicable to the wide area. However, the problems to be solved are divergent: incremental computation tries to reduce the CPU cost of updates. We are concerned with incorporating updates in the presence of limited or sporadic connectivity.

Twitter has built an analytics system atop Cassandra called Rainbird [27]. Rainbird is designed to do near-real-time counting and statistics on large data volumes. This demonstrates the industrial need for the sort of analysis we envision. Rainbird is designed for single datacenter deployment, however, and does not explore widely distributed storage or adaptive control.

6 Conclusions

Many data sources create widely distributed data. Backhauling the data before storing requires culling data too early, and therefore results in suboptimal choices of what to keep and collect. We believe it is time to embrace edge storage and distributed queries instead.

Edge storage will require far-reaching changes to the modern analysis software stack. Without wide-area replication, we need new mechanisms for handling slow and unavailable nodes. We advocate structured storage and degradation policies. OLAP cubes are an ideal abstraction for this, since they are familiar to users, they can be implemented efficiently, and they are sufficiently structured for the system to plan queries in the face of changing network conditions.

"Big data", until now, has predominantly meant data in big centralized datacenters. It is time to build systems for storing and processing widely dispersed data. Doing so will let users get the greatest value from each bit of data.

References

[1] Bringing big data to the enterprise. http://www-01.ibm.com/software/data/bigdata/, 2012.

[2] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. B. Zdonik. The design of the Borealis stream processing engine. In *CIDR*, 2005.

[3] S. Agarwal, A. P. Iyer, A. Panda, S. Madden, B. Mozafari, and I. Stoica. Blink and it's done: interactive queries on very large data. *VLDB*, 5(12), 2012.

[4] Y. Ahmad, O. Kennedy, C. Koch, and M. Nikolic. Dbtoaster: Higher-order delta processing for dynamic, frequently fresh views. *VLDB*, 5(10), 2012.

[5] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, et al. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report 2009-28, UC Berkeley, 2009.

[6] S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, F. Reiss, and M. Shah. TelegraphCQ: Continuous dataflow processing. In *SIGMOD*, 2003.

[7] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[8] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo. Spade: the system s declarative stream processing engine. In *SIGMOD*, 2008.

[9] L. Golab, T. Johnson, J. S. Seidel, and V. Shkapenyuk. Stream warehousing with DataDepot. In *SIGMOD*, 2009.

[10] J. Gray et al. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[11] J. Han, Y. Chen, G. Dong, J. Pei, B. W. Wah, J. Wang, and Y. D. Cai. Stream cube: An architecture for multi-dimensional analysis of data streams. *Distributed and Parallel Databases*, 18(2):173–197, 2005.

[12] J.-H. Hwang, U. Cetintemel, and S. B. Zdonik. Fast and reliable stream processing over wide area networks. In *Data Engineering Workshop*, 2007.

[13] Z. Ives, T. Green, G. Karvounarakis, N. Taylor, V. Tannen, P. Talukdar, M. Jacob, and F. Pereira. The orchestra collaborative data sharing system. *SIGMOD Record*, 37(3), 2008.

[14] S. Jarr. The big data value continuum. http://blog.voltdb.com/big-data-value-continuum/, 2012.

[15] K. Keeton, P. Mehra, and J. Wilkes. Do you know your IQ?: A research agenda for information quality in systems. *SIGMETRICS PER*, 37(3):26–31, 2010.

[16] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny AGgregation service for ad-hoc sensor networks. In *OSDI*, 2002.

[17] F. McSherry, D. Murray, R. Isaacs, and M. Isard. Differential dataflow. In *CIDR*, 2013.

[18] S. R. Mihaylov, Z. G. Ives, and S. Guha. Rex: recursive, delta-based data-centric computation. *VLDB*, 5(11), 2012.

[19] A. Oliner, A. Ganapathi, and W. Xu. Advances and challenges in log analysis. *Comm. ACM*, 55(2), 2012.

[20] B. Panzer. Technology, market and cost trends 2012. https://espace.cern.ch/WLCG-document-repository/Technical_Documents/Technology_Market_Cost_Trends_2012_v23.pdf, 2012.

[21] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. In *ICDE*, 2006.

[22] A. Reisman. Wired bandwidth prices, and what to expect in the future. http://netequalizernews.com/2012/06/27/wired-bandwidth-prices-what-to-expect-in-the-future/, 2012.

[23] T. K. Sellis. Multiple-query optimization. *ACM Trans. Database Syst.*, 13(1):23–52, Mar. 1988.

[24] Storm. https://github.com/nathanmarz/storm/, 2012.

[25] TeleGeography. Telegeography submarine cable map. http://submarine-cable-map-2012.telegeography.com/, 2012.

[26] TeleGeography. Global bandwidth research service executive summary. http://www.telegeography.com/page_attachments/products/website/research-services/global-bandwidth-research-service/0002/8233/gb12-exec-sum.pdf, 2012.

[27] K. Weil. Rainbird: Real-time analytics @twitter. http://www.slideshare.net/kevinweil/rainbird-realtime-analytics-at-twitter-strata-2011, 2011.