Maintaining Coherent Views over Dynamic Distributed Data

Krithi Ramamritham

Computer Science and Engineering Department
Indian Institute of Technology Bombay
krithi@iitb.ac.in

Abstract. Data delivered today over the web reflects rapid and unpredictable changes in the world around us. We are increasingly relying on content that provides dynamic, interactive, personalized experiences. To achieve this, the content of most web pages is created dynamically, by executing queries dynamically, using data that changes dynamically from distributed sources identified dynamically. A typical web page presents a "view" of the world constructed from multiple sources. In this paper, we examine the nature of dynamics of distributed data and discuss fresh approaches to maintain the coherency of the views seen by the users. Achieving such coherency while developing scalable low-overhead solutions poses challenges in terms of delivering data with the required fidelity in spite of data dynamics as well as failures in the infrastructure. How these challenges can be met by the judicious design of algorithms for data dissemination, caching, and cooperation forms the crux of our work.

Keywords: Views, dynamic data dissemination, web, internet scale algorithms, data coherency

The web is becoming a universal medium for information publication and usage. Such information is becoming more and more dynamic and usage is varying from simple tracking to online decision making in real time. Applications include auctions, personal portfolio valuations for financial decisions, route planning based on traffic information, etc. For such applications, data from one or more independent data sources may be aggregated to determine if some action is warranted. Given the increasing number of such applications that make use of highly dynamic data, there is significant interest in systems that can efficiently deliver the relevant updates automatically.

In this paper we summarize our work related to the topic of executing continuous queries over dynamic data so that correct results are always presented to users. Specifically, we present low-cost, scalable techniques to answer continuous aggregation queries using a Content Distribution Network (CDN) of dynamic data items. In such a network of data aggregators (DAs), each data aggregator serves a set of data items at specific coherencies. Just as various fragments of a dynamic web-page are served by one or more nodes of a CDN, serving a query involves decomposing a client query into sub-queries and executing sub-queries on judiciously chosen data aggregators with their individual sub-query incoherency bounds.

Caches and Views. Seen in the Web context, websites have transitioned from a static content model, where content is served from ready-made files, to a dynamic content model, where content is generated on demand. Dynamic content generation allows sites to offer a wider variety of services and content. But, as Internet traffic continues to grow and websites become increasingly complex, performance and scalability are major issues for website owners and users. Web 3.0 techniques provide website visitors with dynamic, interactive, and personalized experiences. However while serving dynamic content, two types of latencies are encountered:

1) Network Latency -- Latency due to communication between users and data sources and numerous nodes in the network (e.g., routers, switches) and
2) Content creation latency -- Latency due to computationally-intensive logic executed at multiple tiers within the sources or within the nodes that lie between the client and the sources.

A widely used existing approach to address WWW performance problems is based on the notion of content caching. These content caching approaches store content at various locations outside the site infrastructure and can improve website performance by reducing content generation delays. Dynamic pages are typically designed according to a template, which specifies layout and content of page wherein pages are decomposed into fragments; fragments and templates are cached at the edge and pages are assembled from fragments, on demand. Alternatively, page layout can itself be generated at run-time [1]. Thus, caching can be done at either (a) the page level, which does not guarantee that correct pages are served and provides very limited reusability, or (b) the fragment level, which is associated with several design level and runtime scalability issues. To address these issues, several back-end caching approaches have been proposed, including query result caching and fragment level caching. In general cached content includes media files (pictures, audio, video) or dynamically generated page fragments; Content generated for one user is saved, and used to serve subsequent requests for the same content. A fragment is a portion of a web page, or at the granularity of a programmatic object. This type of cache works with a dynamic content application to reduce the computational and communication resources required to build the page on the site, thus reducing server-side delays. The holy grail of dynamic content caching is the ability to cache dynamic content at finer granularities outside the site's infrastructure. Such an approach would provide the benefits of caching finer granularities of content (e.g., greater reusability), while simultaneously achieving the benefits associated with proxy-based caching (e.g., reduced bandwidth, reduced firewall processing).

In databases, views are created to avoid repeated execution of sub-expressions that occur in queries. Which views to create and how to maintain them, i.e., make sure that the materialized forms of the views are up-to-date, thereby reflecting the current values of the base relations that contribute to the views, are part of the query optimization problem. In the systems context, cached values represent values evaluated once, but used multiple times. Whereas cache units have historically been physical entities like pages or cache lines, more recently the notion of logical caches has become commonplace, making a cache element similar in scope to a view. In a distributed systems context view maintenance or keeping a cache element up-to-date involves making sure that the "value" or a view is the same irrespective of whether the cached/materialized form of the view is used/seen or the view/cache is constructed at the time it is used/seen. The challenge in ensuring this arises due to the distributed nature of the data sources which in turn implies communicating changes in the sources to the views so that views are up-to-date.

In this paper, we offer a set of techniques for view/cache maintenance wherein

- o instead of invalidating (typically, cache maintenance is invalidation-based since it is simple to just mark a cached element invalid whenever the source has changed), we refresh a cached fragment.
    - by continuous refresh of the cache fragment, the cache/view can always be kept in a usable state.
- o instead of refreshing after each update, we refresh in a use-specific manner.
    - by developing selective refresh mechanisms, overheads of view maintenance can be minimized.
- o instead of a dynamic fragment being refreshed by the source, we use a peer cache's content for refresh.
    - by constructing data dissemination networks data accuracy requirements of a large number of clients can be satisfied in a scalable manner.

Data coherency. Not every update at the source of data *d* leads to a refresh message being sent to a node that caches *d*. This is because, data usage usually can tolerate some incoherency. In most real-world applications, strong coherency, whereby a client and source always are in sync with each other, is neither required nor affordable. Several relaxations of strong coherency have been proposed:

- o Time domain: Δt - coherency
    - The client is never out of sync with the source by more than Δt time units, e.g., traffic data not stale by more than a minute
- o Value domain: Δv - coherency
    - The difference in the data values at the client and the source are bounded by Δv at all times, e.g., user is only interested in temperature changes > 1 degree; user is only interested in changes in traffic load > 100 messages/sec

Formally, let $s_i(t)$ denote the value of the i^{th} data item at the data source at time t. Let the value of the i^{th} data item known to the user be $d_i(t)$. Then the data incoherency is given by $|s_i(t)-d_i(t)|$. For a data item which needs to be refreshed at an incoherency bound C, a data refresh message is sent to the user if data incoherency $|s_i(t)-d_i(t)| > C$.

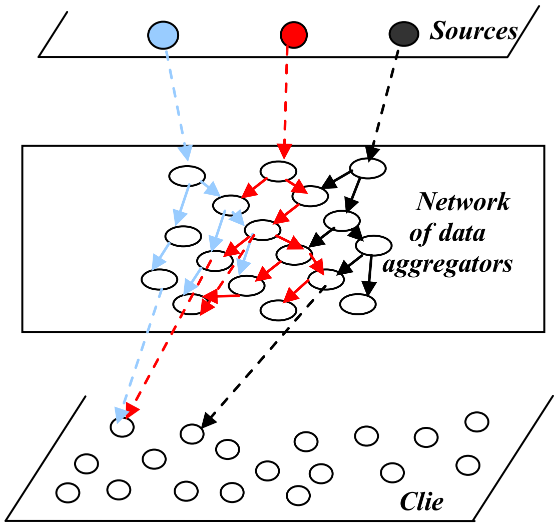
Dynamic content distribution network of data aggregators. Refreshes occur from data sources to the clients through a network of aggregators. In hierarchical data dissemination network, a higher level aggregator guarantees a tighter incoherency bound (i.e., a lower value of C) compared to a lower level aggregator. For maintaining a certain incoherency bound, a data aggregator (DA) gets data updates from the data source or some higher level DA so that the achieved data incoherency is at least as stringent as the specified data incoherency bound. From a data dissemination capability point of view, each DA is characterized by a set of (S_i, c_i) pairs, where S_i is a data item which the DA can disseminate at an incoherency bound c_i.

Each DA in the network can be seen as providing a view over the dynamic data provided by the sources.

Consider a network of data aggregators managing data items S_1-S_4.

Aggregators can be characterized as-
d_1: {(S_1, 0.5), (S_3, 0.2)}
d_2: {(S_1, 1.0), (S_2, 0.1), (S_4, 0.2)}
Aggregator d_1 can serve values of S_1 with an incoherency bound greater than or equal to 0.5 whereas d_2 can disseminate the same data item at a looser incoherency bound of 1.0 or more.

In such a dissemination network of multiple data items all the nodes can be considered as peers since a node d_1 can be child of another node d_2 for a data item S_1 (incoherency bound at d_1 is greater than that at d_2) but the node d_1 can be parent of d_2 for another data item S_2.



Data dissemination network

Given the data and coherency needs of DAs, how should they configure themselves and cooperate to satisfy these needs? How should DAs refresh each other's data such that data coherency requirements are satisfied? Consider the design and building of a dynamic data distribution system that is coherence-preserving, i.e., the delivered data must preserve associated coherence requirements (the user-specified bound on tolerable imprecision) and must be resilient to failures. To this end, we consider a system in which a set of DAs cooperate with each other and the sources, forming a peer-to-peer network. In this system, necessary changes are pushed to the users so that they are automatically informed about changes of interest. In [3] we present techniques 1) to determine when to push an update from one repository to another for coherence maintenance, 2) to construct an efficient dissemination tree for propagating changes from sources to cooperating repositories, and 3) to make the system resilient to failures. We experimentally demonstrate that 1) careful dissemination of updates through a network of cooperating repositories can substantially lower the cost of coherence maintenance, 2) unless designed carefully, even push-based systems experience considerable loss in fidelity due to message delays and processing costs, 3) the computational and communication cost of achieving resiliency can be made to be low, and 4) surprisingly, adding resiliency can actually improve fidelity even in the absence of failures.

Now we move our attention from individual data items to queries over these data items.

Continuous aggregate queries over dynamic data. Consider the following specific applications involving continuous queries used to monitor changes to time varying data and to provide results useful for online decision making. These aggregation queries are long running queries as data is continuously changing and the user is interested in notifications when certain conditions hold. Thus, responses to these queries are to be refreshed continuously, respecting the coherency requirements associated with the queries.

For tracking portfolios, we execute the following types of queries.
- portfolio query involving shares in stock exchanges in a country
  Σ (Number of shares of company i
  × current price of share of company i)
- global portfolio query involving stocks in exchanges in different countries
  Σ (Number of shares of company i
  × current price of share of company i in country j
  × currency exchange rate with country j)

both the stock price in the foreign currency and the currency exchange rate change continuously.

For monitoring network traffic to capture unusual activity such as network bombardment or denial of service attacks, given a condition involving data from certain sources, or certain destinations, each monitoring node evaluates

Σ frequency count of messages satisfying the condition

Finally, consider monitoring dynamic physical phenomena using sensor networks. Suppose a disaster management team is interested in tracking an oil spill with the help of sensors. Sensors track the perimeter of the "circular" spill. Center of the spill (x0, y0) can be approximated by the average of the points on the perimeter. By tracking (xi, yi) points on the perimeter, area under the spill can be calculated by continuously evaluating the expression:

$\Pi ((xj - x0)^2 + (yj - y0)^2)$.

Given a data dissemination network and above queries over the dynamic data contained in the nodes, the following interrelated questions arise:

Given a set of continuous queries at a DA how should we assign accuracy bounds for each data item? Whereas the above work considers networking DAs with specific data, [4] considers the problem of assigning data accuracy bounds to these data items. Assigning data accuracy bounds for data used by non-linear queries poses special challenges. Unlike linear queries, data accuracy bounds for non-linear queries depend on the current values of data items and hence need to be recomputed frequently. So, we seek an assignment such that a) if the value of each data item at C is within its data accuracy bound then the value of each query is also within its accuracy bound, b) the number of data refreshes sent by sources to C to meet the query accuracy bounds, is as low as possible, and c) the number of times the data accuracy bounds need to be recomputed is as low as possible. In [4], we couple novel ideas with existing optimization techniques to derive such an assignment. Specifically, we make the following contributions: (i) Propose a novel technique that significantly reduces the number of times data accuracy bounds must be recomputed; (ii) Show that a small increase in the number of data refreshes can lead to a large reduction in the number of re-computations; we introduce this as a tradeoff in our approach; (iii) Give principled heuristics for addressing negative coefficient polynomial queries where no known optimization techniques can be used; we also prove that under many practically encountered conditions our heuristics can be close to optimal; and (iv) Experimentally demonstrate the efficacy of our techniques in handling large number of polynomial queries.

Given the data and the coherency available at DAs, and a set of continuous queries, how do we plan the query executions, that is, how do we divide the query into sub-queries, and allocate the query incoherency bound among them? In case of a CDN, web page's division into fragments is a page design issue, whereas, for continuous aggregation queries, this issue of dividing a query into sub-queries has to be handled on per-query basis by considering *data dissemination capabilities* of DAs.

Consider a portfolio query $Q1 = 50\, S_1 + 200\, S_2 + 150\, S_3$, with a required incoherency bound of 80 (in a stock portfolio S_1, S_2, S_3 can be different stocks and incoherency bound can be \$80). We want to execute this query over the DAs, minimizing the number of refreshes. There are various options for the client to get the query result:

1) The client may get the data items S_1, S_2 and S_3 separately. The query incoherency bound can be divided among data items in various ways ensuring that query incoherency is below the incoherency bound. In this paper, we show that getting data items independently is a costly option. This strategy ignores the fact that the client is interested only in the aggregated value of the data items and various aggregators can disseminate more than one data item.
2) If a single DA can disseminate all three data items required to answer the client query, the DA can construct a composite data item corresponding to the client query (S_q=50 S_1 + 200 S_2 + 150 S_3) and disseminate the result to the client so that the query incoherency bound is not violated. It is obvious that if we get the query result from a single DA, the number of refreshes will be minimum (as data item updates may cancel out each other, thereby maintaining the query results within the incoherency bound). As different DAs disseminate different subsets of data items, no DA may have all the data items required to execute the client query. Further, even if an aggregator can refresh all the data items, it may not be able to satisfy the query coherency requirements. In such cases the query has to be executed with data from multiple aggregators.
3) Divide the query into a number of sub-queries and get their values from individual DAs. In that case, the client query result is obtained by combining the results of multiple sub-queries. For the DAs given earlier, *Q1* can be divided in two alternative ways:

    Plan1: Result of sub-query 50 S_1 + 150 S_3 is served by d_1 whereas value of S_2 is served by d_2.

Plan2: Value of S_3 is served by d_1 whereas result of sub-query 50 S_1 + 200 S_2 is served by d_2.

In both the plans, combining the sub-query values at the client gives the query result. But, selecting the optimal plan among various options is not-trivial. Intuitively, we should be selecting the plan with lesser number of sub-queries. But that is not guaranteed to be the plan with the least number of messages. Further, we should select the sub-queries such that updates to various data items appearing in a sub-query have more chances of canceling each other as that will reduce the need for refresh to the client. In the above example, if updates to S_1 and S_3 are such that when S_1 increases, S_3 decreases, and vice-versa, then selecting *plan1* may be beneficial. In [2] we give a method to select the query plan based on these observations. While solving the above problem, we ensure that each data item for a client query is disseminated by one and only one DA. Although a query can be divided in such a way that a single data item is served by multiple DAs (e.g., 50 S_1 + 200 S_2 + 150 S_3 is divided into two sub-queries 50 S_1 + 130 S_2 and 70 S_2 + 150 S_3); but in doing so the same data item needs to be processed at multiple aggregators, increasing the unnecessary processing load. By dividing the client query into disjoint sub-queries we ensure that a data item update is processed only once for each query (for example, in case of paid data subscriptions it is not prudent to get the same data item from the multiple sources).

Sub-query incoherency bounds are required to be found using the query incoherency bounds such that, besides satisfying the client coherency requirements, the chosen DA (where the sub-query is to be executed) is capable of satisfying the allocated sub-query incoherency bound. For example, in *plan1*, incoherency bound allocated to the sub-query $50S_1 + 150S_3$ should be greater than 55 (=50*0.5+150*0.2) as that is the tightest incoherency bound which the aggregator d_1 can satisfy. Clearly, the number of refreshes will depend on the division of the query incoherency bounds among sub-query incoherency bounds.

In [2] we provide a technique for getting the optimal query plan (i.e., set of sub-queries with their incoherency bounds and DAs where the sub-queries will be executed) which satisfies client query's coherency requirement with least cost, measured in terms of the number of refresh messages sent from aggregators to the client. For estimating query execution cost, we build a continuous query cost model which can be used to estimate the number of messages required to satisfy the client specified incoherency bound. Performance results using real-world traces show that our cost based query planning leads to queries being executed using less than one third the number of messages required by existing schemes.

Details of our work related to dynamic data dissemination can be found at
http://www.cse.iitb.ac.in/~krithi/ddd.html.

Acknowledgments. I would like to thank all my Masters and Ph.D. students, especially Shetal Shah and Rajeev Gupta, for their contributions to this project.

References

1) D. VanderMeer, A. Datta, K. Dutta, H. Thomas, K. Ramamritham. Proxy-Based Acceleration of Dynamically Generated Content on the World Wide Web. ACM Transactions on Database Systems (TODS) Vol. 29, June 2004.
2) R. Gupta, K. Ramamritham, Optimized Query Planning of Continuous Aggregation Queries in Dynamic Data Dissemination Networks, WWW 2007, Banff, Canada, May 2007.
3) S. Shah, K. Ramamritham, P. Shenoy, Resilient and Coherency Preserving Dissemination of Dynamic Data Using Cooperating Peers, IEEE Transactions on Knowledge and Data Engineering, Vol. 16, No.7, pp 799 - 812, July 2004.
4) S. Shah, K. Ramamritham, Handling Non-linear Polynomial Queries over Dynamic Data, Proc. of IEEE International Conference on Data Engineering (ICDE), April, 2008.