Optimizing the update packet stream for web applications

Muthuprasanna Muthusrinivasan[1] and Manimaran Govindarasu[2]

[1] Google Inc, Mountain View CA 94043, USA
muthup@google.com
[2] Iowa State University, Ames IA 50011, USA
gmani@iastate.edu

Abstract. The Internet has evolved to an extent where users now expect any-where any-time and any-form access to their personalized data and applications of choice. However providing a coherent (seamless) user experience across multiple devices has been relatively hard to achieve. While the *how to sync* problem has been well studied in literature, the complementary *when to sync* problem has remained relatively unexplored. While frequent updates providing higher user satisfaction/retention are naturally more desirable than sparse updates, the steadily escalating resource costs are a significant bottleneck. We thus propose extensions to the traditional periodic refresh model based on an adaptive *smart sync approach* that enables variable rate updates closely modeling expected user behavior over time. An experimental evaluation on a sizeable subset of users of the GMAIL web interface further indicates that the proposed refresh policy can achieve the best of both worlds - limited resource provisioning and minimal user-perceived delays.

Key words: data synchronization, web applications, cloud computing

1 Introduction

The World Wide Web has seen tremendous growth since its early days at CERN [1], and in the past few years has witnessed a steady shift away from the traditional desktop computing paradigm. The rapid emergence of cloud computing [2] has given rise to - *service providers* who build/manage universally accessible, massively scalable, highly reliable compute infrastructure as an utility/commodity, *software vendors* who host their applications in this cloud avoiding capital expenditure and instead paying only for their resource utilization, and *end users* who can now access technology-enabled applications easily without knowledge of the underlying infrastructure. While cloud/service providers [3] benefit from multi-tenancy and economies of scale, the software vendors benefit from *on-demand* access to resources for their SaaS [4] deployments worldwide.

The greatest beneficiaries though have been the end users - the use of open standards, technological convergence [5] and pervasive computing [6] have enabled users to access information through a multitude of devices - instant messaging

through desktop clients, web browsers, mobile phones, interactive TV etc. is now possible. Users can thus access their data in the cloud using any device, at any time, anywhere in the world, and in any desired form, with no restrictions whatsoever. The widespread adoption of these ubiquitous applications is now primarily governed by natural expectations of a *coherent user experience*. In applications where data reconciliation can be quick and easy, seamless on-access device sync would suffice. However, it is untenable for applications requiring longer synchronization cycles or needing a complex update mechanism, due to undesirable user-perceived delays and slow application response times.

The traditional approach towards data reconciliation has been to execute mutually agreed update protocols at regular intervals. However, high frequency updates for devices having low user activity leads to a large number of redundant *null* updates. Such a non-optimal use of resources does not scale well for the cloud managing millions of users each possibly using tens of remote devices. Hence we propose a *smart sync approach* that exploits user behavior (past access patterns) to determine the likelihood of an impending user access to trigger a pro-active update. This would not only consume far fewer resources due to throttling of updates during periods of expected user inactivity, but also provide maximal data coherence across devices due to pro-active data synchronization. In this paper, we analyze a sizeable subset of GMAIL user interactions with the cloud, to determine optimal user behavior models for likelihood estimation (update interval adaptation) and to also study the resulting benefits for the users and the application in the cloud.

2 Related Work

Continuous harmonization of data over time across multiple remote sources, or *data synchronization* [7], has been a well studied topic [8], and has been explored in multiple contexts such as database design [9], secure communications [10], memory architectures [11], distributed computing [12], etc.

The research on data synchronization for ubiquitous computing has proceeded mainly along two lines, namely the *how to sync* and *when to sync* paradigms. The 'how to sync' issue deals with designing optimal data sync protocols that are scalable with network size and the resulting bandwidth/storage considerations. The traditional approach has relied on the use of timestamps [13] for version control, and delta compression [14] for minimal data transmission. Recent approaches relying on robust mathematical [15], information-theoretic [16], and probabilistic [17] techniques have provided greater flexibility.

Also, commercial solutions including Palm's HotSync [18], Microsoft's ActiveSync [19], Nokia's IntelliSync [20], etc. are in wide use today. This proliferation of proprietary sync protocols and their mutual incompatibilities led to

the Open Mobile Alliance [21] and the emergence of a platform-independent open standard known as SyncML [22].

The complementary 'when to sync' issue deals with the formulation of an optimal update policy - good data coherence while consuming fewer resources in the device, the cloud, and the network. The classic 'push vs pull' debate arises in this context [23]. In a *push* model the server (cloud) keeps track of all the connected clients (devices) and triggers updates as soon as they are generated. While it guarantees maximal data coherence and least resource consumption, scalability has been a minor concern due to expensive state maintenance for the dynamically-changing heterogeneous network topology.

In a *pull* model, the different clients (devices) fetch data from the server (cloud) at some pre-defined intervals. While a lower pull frequency leads to delayed data coherence, a higher pull frequency leads to increased resource consumption. Although achieving an optimal threshold has proven to be difficult, most web applications today use periodic polling for updates as it is both light-weight and easily deployed. Recently hybrid push-pull mechanisms [24] exploiting persistent (keep-alive) HTTP connections and reverse-AJAX style asynchronous polling such as COMET [25] and the Bayeux protocol [26] have been proposed. In parallel, tool-kits such as Google Gears [27], Microsoft Sync Framework [28] and others provide the ability to use applications off-line, yet seamlessly synchronizing with the cloud whenever possible.

Our focus here is on continual adaptation of the popular periodic polling technique based on past access patterns to both improve user satisfaction and network/cloud performance. The rest of the paper is organized as follows. Sections 3 and 4 present the system architecture used for analysis and a few key preliminary observations respectively. Sections 5 and 6 then present the proposed adaptive sync techniques and the results of their experimental evaluation respectively.

3 Analysis Framework

GMAIL [29] is a popular cloud-based email solution used by a large number of people world-wide on multiple platforms/devices. As a means of evaluating our proposed smart sync approach, we analyze a sizeable subset of GMAIL user interactions for a five-week duration (25 week-days). In this context, we wish to categorically state that the web access logs used in this experiment were fully anonymized to mask all user and location identifying information with due concern for privacy. Fig. 1 represents the analysis framework used for our experimental evaluation here. All user queries served by the many GMAIL frontend servers have historically been anonymized and per query statistics archived in the Google File System [30]. We mine these per query logs to construct aggregate

per user logs, and run them through a Sawzall/Map-Reduce [31] [32] based pattern extraction engine that generates session signatures in Bigtable [33]. Lastly we design a query replay mechanism that can gather/compute per user statistics based on continual adaptation of the individual session signatures.

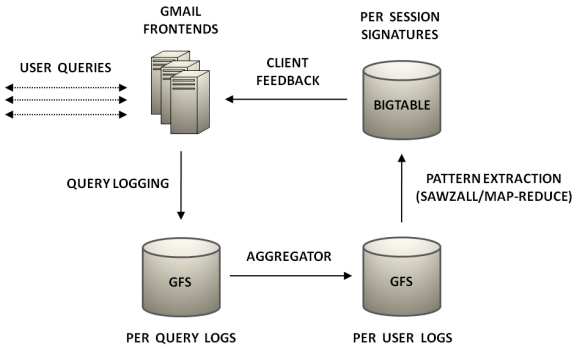


Fig. 1. Analysis Framework

For privacy and confidentiality reasons, our results here primarily focus on frequency distributions and behavioral trends, not absolute statistics and raw numbers. Additionally, the results are expected to have a minor loss of fidelity induced by the anonymization process used to strip sensitive user/location information from the user access logs.

4 Preliminary Study

We now present our initial observations along with a formal definition of a few terms loosely used thus far.

4.1 Terminology

While we expect the terms *user*, *device* and *location* to refer to a person, physical gadget, and some geo-coordinate respectively, their usage in the world of web applications is slightly skewed - every HTTP request is instead represented by a triplet *(user_id, location_id, cookie_id)*.

While *user_id* directly maps a digital identity to a physical person, it is not guaranteed to be unique due to the use of multiple accounts by a single person or sharing of a single account by a group of users. Similarly *location_id* cannot guarantee a unique mapping from a network address to the Internet host that issued that particular request due to the use of network address translation, proxy servers etc. The other means of tracking distinct devices using browser cookies

(*cookie_id*) also cannot guarantee unique device identification due to browser state synchronization across devices by various applications [34] and pro-active user deletion of cookies due to privacy/anonymity concerns or otherwise.

Fortunately these anomalies are relatively fewer and hence *we unify the terms cookie, location and device and use them interchangeably henceforth.* We thus restrict our analysis here to appropriate (user_id, cookie_id) tuples as representative of true user migration patterns on the Internet as a whole.

4.2 Initial Observations

As previously stated, data sync techniques relying on a client pull design require server polling at regular intervals. However the ever-increasing bandwidth needs coupled with higher demand for cpu, memory and disk resources mandate the design and use of a more efficient sync mechanism. Our key observation in this regard has been that most users display fairly stereotypical intra-day access patterns in addition to regular everyday usage, making them fairly repetitive and their access patterns relatively easy to predict.

User Persistence: A critical requirement of any adaptive technique is the availability of sufficient per-user data over extended durations of time to make reasonably accurate predictions. Fig. 2 shows the user persistence histogram for the 25-day window. The x-axis represents the total duration (in days) an user accessed the GMAIL web interface, while the (log-scale) y-axis tracks the total number of users in each of those intervals. Herein we notice the large number of highly persistent (heavy) users (long-tail) which provides sufficient scope for reliable data gathering and analysis, for accurate user behavior predictions.

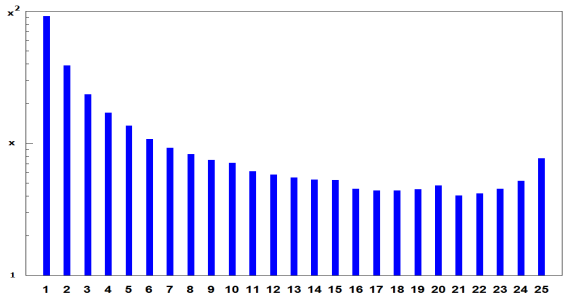


Fig. 2. User Persistence

Device Spread: The user persistence metric measures the time spread, but not the location/device spread for any user. The device spread histogram in Fig. 3 plots the maximum/average/minimum number of cookies accessed by the user during the 25-day observation window. The x-axis represents the total number of

distinct cookies, while the (log-scale) y-axis tracks the number of users accessing the corresponding number of cookies. Herein we notice a large number of users who interact with multiple cookies/devices on a regular basis. The seemingly large number of cookies for certain users is probably due to extensive account sharing or incorrect query to device attribution as explained previously.

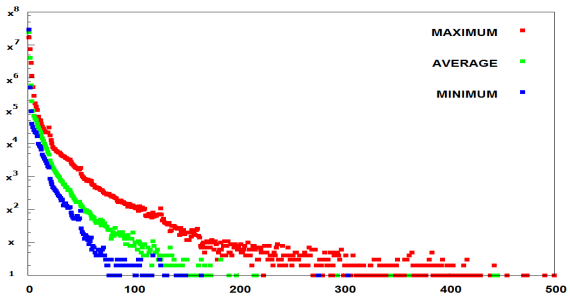


Fig. 3. Device Spread

Time Persistence: Varying the data sync update interval yields higher resource savings for mostly idle users than always-on users, and hence the practical feasibility of any adaptive sync technique is impacted by the cumulative time duration an user accesses a particular session. Fig. 4 represents the time persistence histogram where the x-axis represents the time persistence intervals (in minutes/day), while the (log-scale) y-axis tracks the number of users in each of those intervals. We notice that a majority of users show activity for less than a few hours a day, and hence a periodic polling strategy would perform poorly. The presence of user accounts having extremely high time persistence can be attributed to coarse-grained time data available and mismatched query to user attribution, both induced by content stripping for logs anonymization.

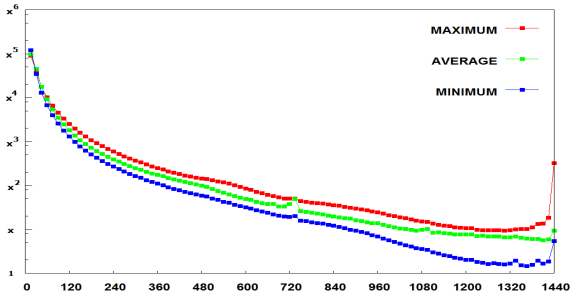


Fig. 4. Time Persistence

Thus high user persistence and device spread combined with low time persistence for a large majority of GMAIL users indicates that an adaptive sync approach

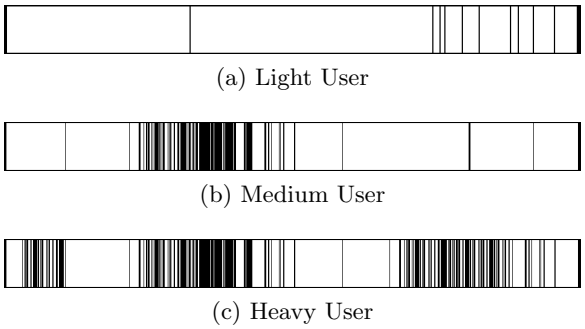
can easily outperform the periodic polling mechanism. We now seek to analyze in greater detail these stereotypical trends in daily user access patterns to design an optimal data refresh policy.

5 Smart Sync Approach

We have thus far focused on the basic *can we adapt* issue, while we now try to address the more critical *how to adapt* issue by analyzing a few user behavior models for simplicity, scalability, and impact on overall user satisfaction.

5.1 Basic Representation

As stated previously, we use a *(user_id, cookie_id)* tuple as the basic unit of data measurement, analysis and subsequent optimization - henceforth simply referred to as a *session*. A steady increase in the number of users and their personal devices can quickly lead to scalability issues with respect to storage and computational needs, and hence we need small light-weight *session signatures* that lend well to easier generation and real-time adaption. Thus complex pattern analysis or correlation techniques need to be discarded in favor of simple resource-efficient approximate solutions.



(a) Light User

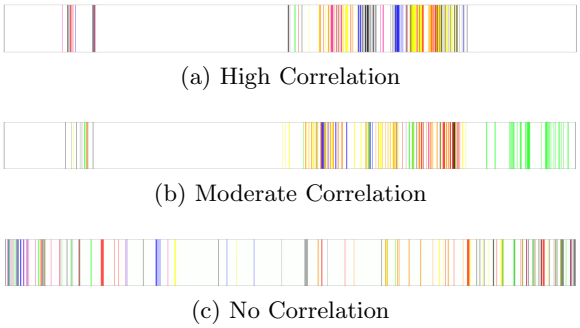
(b) Medium User

(c) Heavy User

Fig. 5. User Session Activity (1 day)

Click Tracking: A simple means of characterizing session activity would be to analyze user-generated clicks in any browser session - while a single click indicates user presence, a corresponding absence may not accurately differentiate an inactive session from an idle user processing related/other information. However, if we presume higher likelihood of an user click in the close neighborhood of a previous click than farther away from it, it provides an indirect means of session state inference - the terms *idle* and *inactive* now assume a more continuous interpretation, lower (higher) deviations indicate idle (inactive) behavior respectively.

In Fig. 5, we partition user session activity patterns into three categories. While Fig. 5a and Fig. 5c represent the two extremes of daily user click activity, Fig. 5b better represents a typical user behavior over a 24-hour period. In Fig. 6, we plot corresponding user activity patterns across multiple days, each day with a different color. While Fig. 6a shows an user having a very high correlation across days, Fig. 6b indicates a moderate level of correlation having a few outliers. On the other hand, Fig. 6c represents an user showing practically no correlation across days. The higher the correlation measured, the more reliable are the resulting user behavior models, and hence better the user click predictions.



(a) High Correlation

(b) Moderate Correlation

(c) No Correlation

Fig. 6. User Session Activity (multiple days)

Click Probability Tracking: Representing a session signature as the union of multiple user click instances on a daily basis leads to variable-sized session signatures. Thus we not only require multiple session signatures for high-activity users, but also larger session signatures due to the higher number of clicks generated. An alternate fixed-size representation that tracks the probability of an user click across multiple contiguous yet disjoint time intervals can alleviate this problem. The session signature size is now determined by the granularity of the individual time intervals and also the precision of the click probabilities in each time interval, and can thus be easily provisioned for better scalability.

Finally, we propose a simple *exponentially weighted moving average* technique to succinctly capture user access patterns across multiple days. Eqn. 1 now represents the session signature (S_n) as the union of multiple user click probabilities (P_n^t) for the different time instances - where α is the weight assigned to any new entry p^t with respect to the previously measured user click probability (P_{n-1}^t) in that time interval (t) but in the older session signature (S_{n-1}).

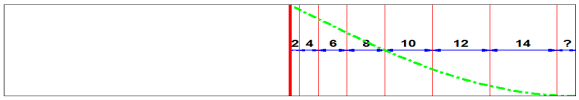
$$S_i = \bigcup_{\forall t} P_i^t = \bigcup_{\forall t} (\alpha * p^t + (1 - \alpha) * P_{i-1}^t) \tag{1}$$

We now extend this basic representation to formulate precise/imprecise models based on individual click probabilities for optimal update interval adaptation.

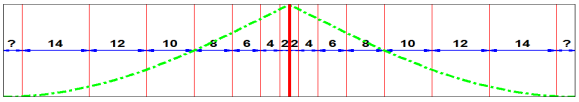
5.2 Click Tracking: Precise Models

We now propose three simple adaptive sync models each displaying a different update interval adaptation around any *single user click*. We then discuss how these per-click models can be easily combined across multiple user clicks both intra-day and inter-day, thereby achieving better resource utilization.

(a) Uniform Refresh (or Basic Periodic Polling)



(b) Throttle Refresh



(c) Predict Refresh

Fig. 7. Per-Click Sync Models

Uniform Refresh: This model assumes an uniform access probability distribution around any user click, and represents the basic periodic polling technique. Fig. 7a thus represents the uniform refresh (periodic polling) at some user click, while Eqn. 2 depicts the individual time instances of subsequent updates, where U_0 represents the registered user click, and Δ is a constant (say, 2 time units).

$$U_n = U_{n-1} + \Delta = U_0 + n * \Delta, \quad \forall\, n > 0 \tag{2}$$

Throttle Refresh: This model assumes a monotonically decreasing access probability distribution around any user click, and hence the refresh interval also steadily increases. Our experimental evaluation of refresh interval adaptation indicates that an arithmetic growth models user access trends more closely than a geometric/exponential growth. Fig. 7b thus represents the throttle refresh at some user click and the decaying access probability, while Eqn. 3 depicts their individual time instances following an additive growth model.

$$U_n = U_{n-1} + n * \Delta = U_0 + \frac{n * (n+1)}{2} * \Delta, \quad \forall\, n > 0 \tag{3}$$

Predict Refresh: This model accounts for refresh interval adaptation not only leading away from the user click, but also leading towards that user click. Thus

it supports pro-active updates for better data coherence, and represents an axial reflection of throttle refresh about the user click. Fig. 7c thus shows the access probability ramp up leading towards and ramp down leading away from some user click, while Eqn. 4 represents the (additive) modulation of the corresponding refresh time instances around the user click.

$$U_n = \begin{cases} U_{n+1} + n * \Delta = U_0 - \frac{n*(n-1)}{2} * \Delta \text{ if } n < 0 \\ U_{n-1} + n * \Delta = U_0 + \frac{n*(n+1)}{2} * \Delta \text{ if } n > 0 \end{cases} \quad (4)$$

The global sync schedule for any session can now be viewed as the interference pattern of the individual sync schedules at each user click, smoothed across multiple days using a simple exponentially weighted moving average technique. It is important to note that we can obtain these precise adaptive sync models only for users having a high click-behavior correlation across multiple days. For users having lower click-behavior correlation, and deployments with limited storage capacity or requiring greater privacy, these models tend to be insufficient.

5.3 Click Probability Tracking: Practical Models

To alleviate privacy concerns surrounding explicit click tracking, we now propose imprecise yet more practical models based on aggregate click probability tracking across multiple contiguous yet disjoint time intervals. In any session, predicting the exact time instant of any user click is infeasible and dependent on myriad factors including the number of emails received, their relative priorities, average user response times etc. Not so surprisingly, it is not infeasible to determine the user click probabilities over longer time intervals - longer the time interval, more precise the click predictions can be. However these longer time intervals render click predictions virtually useless as they cannot guarantee better data coherence (by update pre-fetching) due to the very size of the intervals themselves. Thus we need appropriately-sized time intervals that provide good click predictions (better data coherence), while limiting the session signature storage overhead.

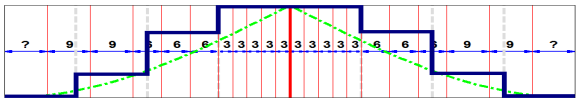
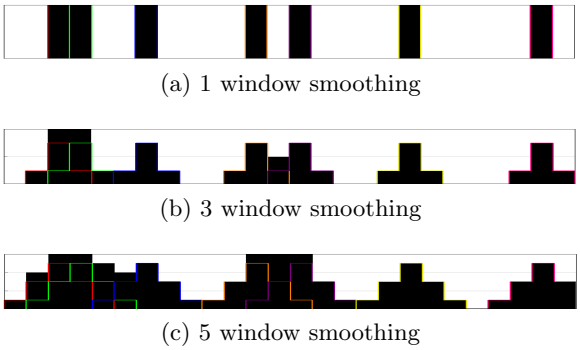


Fig. 8. Slotted Refresh Sync Model

Slotted Refresh: This model is similar to the *predict refresh* model discussed previously, but assumes a discrete probability distribution across the many time interval boundaries. This results in a non-continuous (step-like) growth in refresh intervals, increasing as we traverse time interval boundaries away from an user click. Fig. 8 now represents this slotted refresh model with an additive growth rate (3 time units) across the different static time intervals (15 time units).

Idle Behavior: We introduce another critical parameter here known as the *window/probability spread factor* that accounts for expected idle time between user clicks. User activity typically spans multiple neighboring time intervals wherein not all of them register an user click everyday - these sandwiched zero-activity time intervals thus represent idle user behavior. We now propose a few probability smoothing functions each differently scaling user access probabilities across multiple neighboring time intervals to accurately model these *passively active* idle time intervals. Fig. 9a represents a session signature with distinct click probabilities for each of the user clicks. In Fig. 9b (Fig. 9c) we assume that the idle behavior extends to one (two) time intervals on either side of the user click (scaled probability value). It is critical to note that while a low probability spread results in increased sync lag during idle times, a higher spread leads to gradual probability homogenization and uniform click probabilities across the entire time spectrum, and an appropriate window size is hence critical.



(a) 1 window smoothing

(b) 3 window smoothing

(c) 5 window smoothing

Fig. 9. Probability Smoothing: Window Spread Factor

Finally, the *global sync schedule* can again be viewed as the interference pattern of the individual sync schedules wherein the click probability in any time interval is the sum of the individual (scaled) click probabilities as determined by the window spread factor of its neighboring user click regions. As discussed above, a simple exponentially weighted moving average technique can once again be used to smooth the effect of variations in user behavior across multiple days.

To summarize, we have thus far modeled session signatures as a collection of contiguous yet disjoint time intervals, wherein each unit represents the user access probability (and hence the associated sync lag) for that time interval. We have also sought to detect idle user behavior and subsequently employed a smoothing function to guarantee better data coherence and thus higher user satisfaction. We now seek to individually evaluate both the precise and the imprecise adaptive sync models against a large number of GMAIL user activity streams to study their relative benefits and performance characteristics.

6 Experimental Evaluation

We now analyze both the per-click adaptive sync models and the more practical slotted refresh model by experimental evaluation on a large sample of GMAIL user activity streams. We mainly focus on two critical metrics, resource utilization with respect to the total number of refresh queries issued by any client device/session, and user satisfaction based on the instantaneous data sync lag experienced by the users. In this context, we assume that every refresh query consumes a fixed amount of resources in the client (device), the network, and the server (cloud), to simplify the amortized cost calculations.

6.1 Click Tracking: Precise Models

The per-click models capture the total automated refresh query overhead associated with any user click - we compare the traditional periodic polling (uniform refresh) model against the proposed throttle/predict refresh models here. For sparse user click distributions (Fig. 5a) the growth rate is roughly linear in the total number of uncorrelated user clicks, while it is largely sub-linear for a denser distribution (Fig. 5b), as explained previously.

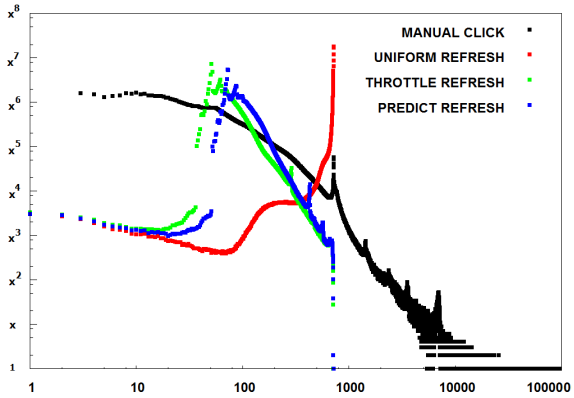
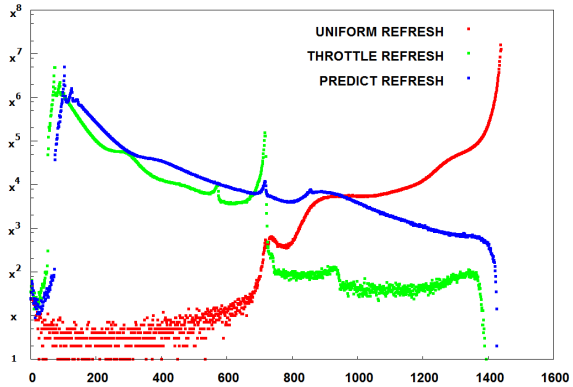


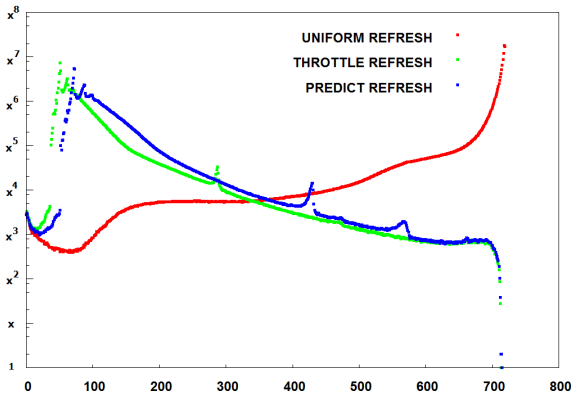
Fig. 10. Query Histogram: Manual vs Refresh Queries (2 minute intervals)

Resource Utilization: Fig. 10 shows the query histogram for all GMAIL users measured over a 24-hour period. The (log-scale) x-axis represents the total number of queries, while the (log-scale) y-axis represents the number of users that generated the corresponding number of queries. We notice that the manual clicks closely follow a *power law* distribution with very few users showing very high user activity (Fig. 5c). Figs. 11a, 11b now depict the same background refresh query trends for the three per-click models with a linear scale x-axis and different minimum refresh intervals. The uniform refresh curve with a positive slope provides a non-scalable model to support an ever-expanding user base and emerging trends

in higher device spread (Fig. 3) and lower time persistence (Fig. 4). On the other hand, the throttle and predict refresh curves with a negative slope provide a more scalable model, with fewer users requiring a high number of background queries. While the predict refresh curve closely tracks the throttle refresh curve for the longer interval here, it shows a greater divergence for the shorter interval - this strange behavior can only be attributed to the specific user click distributions and the refresh interval durations analyzed, and displays high variance across days to postulate any reasonable generalization of this trend.



(a) Refresh @ 1 minute intervals



(b) Refresh @ 2 minute intervals

Fig. 11. Query Histogram: Per-Click Sync Models (varying intervals)

User Satisfaction: We now measure content sync lag at any user click instance as a means of measuring user satisfaction or data coherence. A lower data sync lag provides greater data coherence, and hence better user satisfaction. While the uniform and predict refresh models consume more resources, they bound the maximum (user-perceived) sync lag to pre-determined values, and hence provide

greater control over data coherence. On the other hand, the throttle refresh model provides greater resource savings at the cost of potentially unbounded sync lag - not only across different sessions, but also across user clicks within a single session. Fig. 12 now shows the (throttle refresh) delay histogram for all user sessions, the x-axis representing the sync lag (in minutes) and the (log-scale) y-axis representing the number of users experiencing the corresponding sync lag. We see that the shorter interval refreshes naturally provide a better data coherence, and also that very few users experience a large sync lag. While the mean and standard deviation of sync lags display a wide spread, their average values across all user sessions is of the order of 2-4 minutes at best, and thus provides maximal user satisfaction.

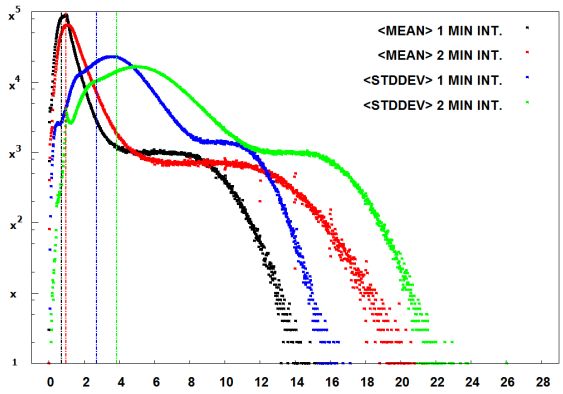


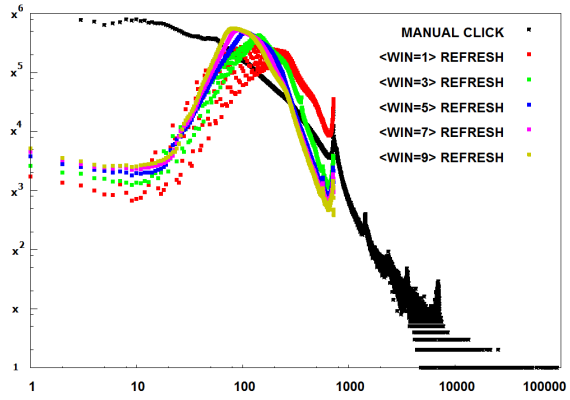
Fig. 12. Throttle Refresh: User-Perceived Sync Lag

Thus the throttle refresh model providing no guarantees on maximum sync lag does a reasonably good job of maintaining good data coherence. To summarize, while the uniform and predict refresh models provide fine-grained control over user-perceived data sync lag, the throttle refresh model behaves more like a best effort model with a few outliers. In short while throttle refresh would suffice for casual users, it might be prudent to employ the predict refresh model for the business or power users - the extra incentive of a relatively smaller data sync lag being delivered at slightly higher resource costs for the service provider.

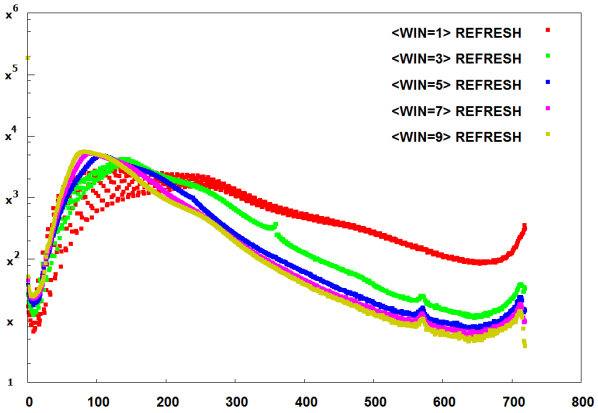
6.2 Click Probability Tracking: Practical Models

For our analysis here, we aggregate daily user click activity into 15-minute time intervals - each session signature thus tracks 96 independent user access probability values. In order to quickly discard spurious one-off user sessions and to efficiently manage the storage requirements for caching multiple user session signatures, we propose limits on the number of session signatures being tracked on a per-user basis using an *adaptive replacement cache (ARC)* [35] policy.

Resource Utilization: Fig. 13a shows the query histogram for all GMAIL users for the different window spread factor sizes, where the (log-scale) x-axis and (log-scale) y-axis represent the number of queries and the number of users that generated the corresponding number of queries respectively. We notice that the background refresh queries here show a similar negative-slope as the per-click models discussed previously. In Fig. 13b we notice the slight shift in the waveform shape as the window spread factor size increases. This result can be interpreted as follows - while the higher window spread factor size does increase the click probability in neighboring time intervals, it also scales down the relative click probability in the current time interval. The net effect of the inter-day exponentially weighted moving average smoothing and that of the intra-day probability scaling (window spreading) in this case yields lower update pre-fetches as the window spread factor increases.



(a) Manual vs Refresh Queries



(b) Refresh Queries (different window sizes)

Fig. 13. Query Histogram: Practical (Slotted Refresh) Model

In order to determine the optimal window spread factor for any user given its many indirect dependencies as listed above, we adopt the *periodic window spread tightening* concept here. We bootstrap with a high probability spread, and then relatively tighten the adapted click probability distribution every few weeks - the lower the session signature correlation across multiple days, the lesser is the window spread tightening provided and vice versa (Fig. 14). While this issue merits an in-depth discussion in itself, we do not delve into greater details here due to space constraints. We suffice it to mention that while this optimization is optional, it does provide incrementally higher benefits.

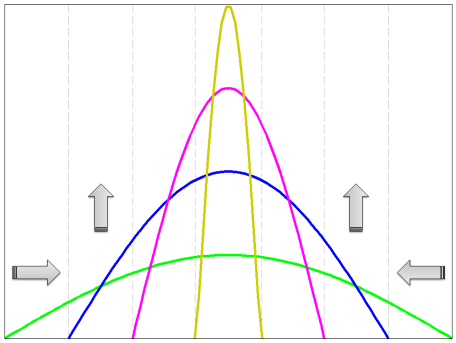


Fig. 14. Probability (Window) Spread Tightening Concept

User Satisfaction: We now measure the effect of the different window spread factor sizes on user-perceived data sync lag. Fig. 15 shows the delay histogram for all user sessions, the x-axis representing the sync lag (in minutes) and the (log-scale) y-axis representing the number of users experiencing the corresponding sync lag. We see that better data coherence is achieved for smaller window spread factor sizes, and this can similarly be explained on the basis of relative probability scaling as discussed above. While the mean and standard deviation of sync lag displays a wide spread, their average values across all user sessions is less than a minute - thereby achieving maximal user satisfaction levels.

To summarize, the proposed data sync models vastly outperform the naive periodic polling mechanism - amongst the many adaptive sync models proposed, the slotted refresh model provides the most practical means of achieving better resources utilization and higher user satisfaction. We also wish to state that the *one-size fits all approach* may not work here - different cloud-based applications might benefit from vastly different models and appropriate choices can only be made by analysis of the individual application requirements and their corresponding user traffic patterns. In this context, we wish to again state that a vastly superior but complex (not cheap) model may not be the most prudent choice with respect to scalability concerns and the resulting computation costs.

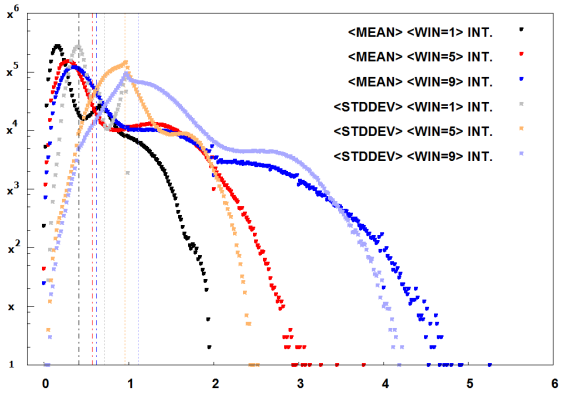


Fig. 15. Slotted Refresh: User-Perceived Sync Lag

7 Conclusions

Web applications today have greatly evolved to provide users instantaneous access to their personalized data from any device/location and in any form with extreme ease. While users today have come to expect seamless data migration across their many devices, various technological constraints with respect to high data coherence and better data synchronization do impose certain barriers. The traditional approach of periodic data fetch by the clients from the different servers in the cloud faces critical limitations with respect to scalability and prohibitively high costs. We thus seek to address the dual-fold problem here - increased user satisfaction and resource savings - by proposing novel extensions to the popular periodic polling technique.

The focus of this work is on addressing the classic trade-off between resource utilization and user satisfaction. Higher the data refresh rate, higher the costs, but also higher the user satisfaction. Conversely, lower the data refresh rate, lower the costs, and correspondingly lower the user satisfaction. The opportunity thus lies in identifying the sweet spot along the entire spectrum that achieves the right balance between the two metrics. Our approach here has been to understand individual user behavior based on past access patterns and thereby derive future predictions for user access with high confidence, so that preemptive/throttled data sync can be easily achieved. An experimental evaluation of a large sample of GMAIL user activity streams successfully validates our approach, and we now plan to perform limited user trials hopefully leading to wide-spread deployment.

References

1. Website of the world's first-ever web server, *http://info.cern.ch*
2. Cloud Computing, *http://en.wikipedia.org/wiki/Cloud_computing*

3. A guided tour of Amazon, Google, AppNexus, *http://www.infoworld.com/article/08/07/21/30TC-cloud-reviews_1.html*
4. SaaS, *http://en.wikipedia.org/wiki/Software_as_a_service*
5. Convergence, *http://en.wikipedia.org/wiki/Media_convergence*
6. Pervasive Computing, *http://en.wikipedia.org/wiki/Pervasive_computing*
7. A. Pikovsky, M. Rosenblum, J. Kurths, *Synchronization: A Universal Concept in Nonlinear Sciences*, Cambridge University Press, 2001
8. W. Kautz, *Fibonacci codes for synchronization control*, IEEE Trans. on Information Theory, 11(2):284-292, 1965
9. J.J. Kistler, M. Satyanarayanan, *Disconnected Operation in the Coda File System*, ACM TOCS, 10(1):3-25, 1992
10. T. L. Liao, S. H. Tsai, *Adaptive synchronization of chaotic systems and its application to secure communications*, Chaos, Solitons, Fractals, 11(9):1387-1396, 2000
11. J. Mellor-Crummey, M. Scott, *Algorithms for scalable synchronization on shared-memory multiprocessors*, ACM TOCS, 9(1):21-65, 1991
12. H. Kopetz, W. Ochsenreiter, *Clock synchronization in distributed real-time systems*, IEEE Trans. on Computers, 36(8):933-940, 1987
13. P. Bernstein, N. Goodman, *Timestamp-based algorithms for concurrency control in distributed database systems*, VLDB, pp. 285-300, 1980
14. RSync, *http://en.wikipedia.org/wiki/Rsync*
15. Y. Minsky, A. Trachtenberg, R. Zippel, *Set reconciliation with nearly optimal communication complexity*, IEEE Trans. on Info. Theory, 49(9):2213-2218, 2003
16. D. Starobinski, A. Trachtenberg, S. Agarwal, *Efficient PDA synchronization*, IEEE Trans. on Mobile Computing, 2(1): 40-51, Jan-Mar 2003
17. L. Fan et.al., *Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol*, IEEE/ACM ToN, 8(3):281-293, 2000
18. Palm HotSync, *http://www.palm.com/us/support/hotsync.html*
19. Using ActiveSync, *http://www.microsoft.com/windowsmobile/en-us/help/synchronize/default.mspx*
20. Nokia IntelliSync, *http://europe.nokia.com/A4164024*
21. Open Mobile Alliance, *http://www.openmobilealliance.org*
22. SyncML, *http://www.syncml.org*
23. E. Bozdag, A. Mesbah, A. V. Duersen, *A Comparison of Push and Pull Techniques for Ajax*, CoRR 2007, *http://arxiv.org/abs/0706.3984*
24. E. Bozdag, A. V. Duersen, *An Adaptive Push/Pull Algorithm for AJAX Applications*, AEWSE, pp. 95-100, 2008
25. Low Latency Data for the Browser, *http://alex.dojotoolkit.org/?p=545*
26. Bayeux Protocol, *http://svn.xantus.org/shortbus/trunk/bayeux*
27. Google Gears, *http://gears.google.com/support*
28. Sync Framework, *http://msdn.microsoft.com/en-us/sync*
29. The official GMAIL Blog, *http://gmailblog.blogspot.com*
30. S. Ghemawat, H. Gobioff, S. Leung, *The Google File System*, ACM SOSP, 2003
31. R. Pike, S. Dorward, R. Griesemer, S. Quinlan, *Interpreting the Data: Parallel Analysis with Sawzall*, Scientific Programming Journal, 13(4):277-298, 2005
32. J. Dean, S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, OSDI, pp. 137-150, 2004
33. F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, R. Gruber, *Bigtable: A Distributed Storage System for Structured Data*, OSDI, pp. 205-218, 2006
34. Mozilla Weave, *http://labs.mozilla.com/projects/weave*
35. Cache Algorithms, *http://en.wikipedia.org/wiki/Cache_algorithms*