DECT: Distributed Evolving Context Tree for Understanding User Behavior Pattern Evolution

Industrial Paper

Xiaokui Shu*
Department of Computer Science, Virginia Tech
Blacksburg, VA USA
subx@cs.vt.edu

Nikolay Laptev
Yahoo! Labs
701 First Avenue
Sunnyvale, CA USA
nlaptev@yahoo-inc.com

Danfeng (Daphne) Yao
Department of Computer Science, Virginia Tech
Blacksburg, VA USA
danfeng@cs.vt.edu

ABSTRACT

Internet user behavior models characterize user browsing dynamics or the transitions among web pages. The models help Internet companies improve their services by accurately targeting customers and providing them the information they want. For instance, specific web pages can be customized and prefetched for individuals based on sequences of web pages they have visited. Existing user behavior models abstracted as time-homogeneous Markov models cannot efficiently model user behavior variation through time. This paper presents DECT, a scalable time-inhomogeneous variable-order Markov model. DECT digests terabytes of user session data and yields user behavior patterns through time. We realize DECT using Apache Spark. Our implementation is being open-sourced and we deploy DECT on top of Yahoo! infrastructure. We demonstrate the benefits of DECT with anomaly detection and ad click rate prediction applications. DECT enables the detection of higher-order path anomalies that are masked out by existing models. DECT also provides deep insights into ad click rates with respect to user visiting paths.

Keywords

Markov Model; Context Tree; Distributed Computing; Time Series; Anomaly Detection; Link Prediction

1. INTRODUCTION

Understanding Internet user behavior is the key to the optimization of Internet information feeding systems. A web page can be prepared/prefetched for a user if the service provider knows the user will visit the page in the short future [23]. Links/ads on a web page can be customized if the service provider understands which links/ads the user is likely to click [17]. Search engines can be designed to fit human browsing dynamics [13].

Markov model (first-order, time-homogeneous) is commonly adopted for Internet user behavior modeling [5]. It is, however, amnesiac; the probability of the next user visit is purely based on the current status of the user. Higher-order Markov models cure the amnesia issue by digesting historical visiting sites of users [15]. Variable-order Markov models improve higher-order Markov models by pruning away unnecessarily higher-order paths for space saving purposes [3].

While the community has developed a string of advanced Markov models to describe Internet user behavior patterns, one strong assumption is constantly kept in all existing models: user behavior patterns do not change over time.

The above assumption, however, does not hold in the real world. New products are releasing; UI of existing web sites/pages are changing; cyber attacks occur; breaking news happen. *The Internet is evolving, and the observed Internet user behavior patterns should reflect the changes.*

This paper presents DECT (distributed evolving context tree), a time-variant model for efficiently describing Internet user behavior patterns and their changes through time. DECT is a time-inhomogeneous variable-order Markov model. It improves the state of the art variable-order Markov models by releasing its assumption of static time-invariant user behavior patterns. DECT is designed to handle large volumes of user session data and can be efficiently constructed via distributed computing.

Time-variant variable analysis, e.g., visit counts of services, has been widely used in industry to detect anomalies like attacks, failures, and bugs. However, these commonly used variables are stateless or only first-order with respect to Markov models.

In contrast, DECT enables higher-order time-variant visiting path analysis. DECT yields both regular time series of individual path visiting probabilities and high-dimensional time series for a set of related paths, e.g., paths that share the same prefix. We demonstrate in Section 4.1 that DECT can produce deep signals for anomaly detection. It helps reveal stealthy attacks, e.g., application layer DDoS attack [21] and browsing mimicry attack [22]. First-order Markov models, in contrast, could mix these signals into noises. We demonstrate in Section 4.2 that DECT distinguishes ad click probability variations based on historical web pages/sites a user visits, while existing first-order prediction is blind to

*The work was done while the first author was an intern at Yahoo! Labs.

Table 1: Symbols, Terms and Definitions

	Term	Definition
s	site	the primitive unit to record user behavior (e.g., a URL, a web service, a website)
E	session	a sequence of sites that a user visits (every session has a beginning and an end)
$\bar{p}$	path	a substring of a session
τ	target	the next site a user is going to transit to
$\bar{c}$	context	a sequence of visited sites prior to τ

different types of users who come from diverse paths.

The contributions of our work are summarized as follows.

- We design DECT to digest large volumes of user session data and construct time-variant user behavior models in a distributed manner.
- We explain the benefits of a time-variant user behavior model and showcase application examples of DECT in anomaly detection and ad click prediction.
- We realize DECT using Apache Spark and demonstrate its performance processing terabytes of real-world user session datasets.

2. DECT

We discuss two major features of DECT in this section: *variable-order* and *time-inhomogeneity*. The former is realized through a *flattened context tree*, and the latter is accomplished through a window sliding process.

2.1 Definitions and Overview

We study user behavior in terms of their visiting paths on the Internet. A (desktop or mobile) user session is recorded as a sequence of visits to a set of Internet *sites* – resources that users are visiting. Sites vary from specific URLs to domains[1]. We define related variables that are used to express user visiting paths in Table 1.

Higher-order Markov models have been proved effective in modeling static user behavior [5]. Given a path $\bar{p}$, a higher-order Markov model can be trained to predict the last transition of $\bar{p}$ based on previously visited sites in $\bar{p}$. In this setup, we refer to the last transition in $\bar{p}$ as the target τ, and the sites prior to τ as the context $\bar{c}$.

The key question we aim to answer is how target transition probabilities change over time. When considering the higher-order Markov model as a weighted directed graph $G_M = (V_M, E_M)$, we construct our model to keep track of:

- change of V_M: new and obsoleted nodes
- change of E_M: transition matrix variation

DECT enables the tracking of both changes, and it provides two features to handle large amounts of data and mitigate exponential space explosion caused by regular higher-order Markov model:

i) variable-order context-target probability
ii) fine-grained parallel path computing and pruning

We realize the two features through *flattened context tree* – a new parallel and concise data structure for building distributed time-inhomogeneous variable-order Markov model.

[1]The granularity of sites is a data collection parameter.

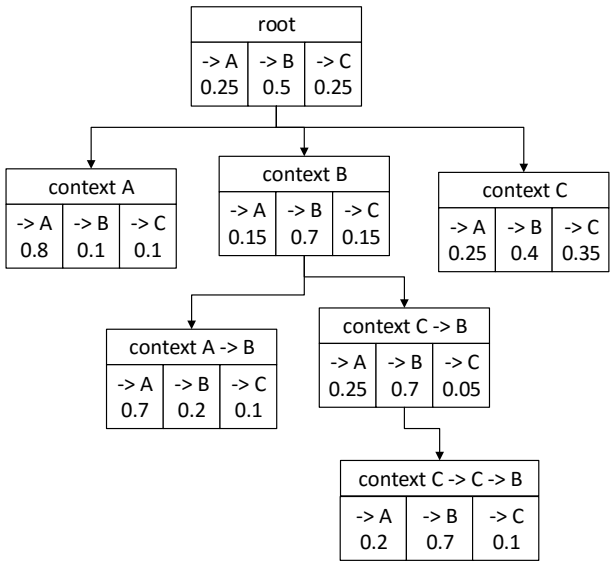

Figure 1: Example of regular context tree (pruned to represent a variable-order Markov model).

2.2 A New Context Tree Structure

Context tree is a common data structure for constructing variable-order Markov model. We first give a brief description of regular context tree and its operations. Then we present our flattened context tree structure for distributed tree construction and fine-grained parallel pruning.

2.2.1 Regular Context Tree

A regular context tree T_C is a k-ary tree where k is the number of all possible sites. T_C records static transition probabilities of any target given a limited length context. The limited context length is the depth of the tree.

We give an example of a regular context tree in Figure 1. A site $s \in \{A, B, C\}$. Each node maps to one context that is recorded. A node $n_{\bar{c}}$, which corresponds to context $\bar{c} = (s_{-y}, \ldots, s_{-2}, s_{-1}, s_0)$, positions at depth y in T_C. Its parent is the node with context $\bar{c}' = (s_{-(y-1)}, \ldots, s_{-2}, s_{-1}, s_0)$ at depth $y-1$. Its children are nodes with context $\bar{c}_{c_i} = (s_{-(y+1)_i}, \ldots, s_{-2}, s_{-1}, s_0)$ at depth $y+1$ where $0 \leq i \leq \xi_{\bar{c}} \leq k$. $\xi_{\bar{c}}$ is the total number of children of $\bar{c}$.

$n_{\bar{c}}$ stores the target probability distribution with respect to the context $\bar{c} = (s_{-y}, \ldots, s_{-2}, s_{-1}, s_0)$, i.e., $P(\tau_j|\bar{c})$ where $0 \leq j \leq \kappa_{\bar{c}} \leq k$. $\kappa_{\bar{c}}$ is the total number of reachable targets given the context $\bar{c}$.

Pruning Pruning a regular context tree of a higher-order Markov model results in a variable-order Markov model. The standard T_C pruning strategy is a bottom-up process: pruning away $n_{\bar{c}}$ if both criteria are satisfied:

1. $n_{\bar{c}}$ is a leaf node.
2. The distance, e.g., KL divergence, between the target probability distribution of $n_{\bar{c}}$ and that of its parent node $n_{\bar{c}'}$ is less than a predefined threshold $\mathcal{T}_{pv}$.

Time-variant capability A regular context tree is designed to accommodate static transition probabilities. Updating the tree to reflect a time-variant model is expensive. It requires to recalculate the probabilities and reevaluate the previous pruning procedures for pruned nodes.

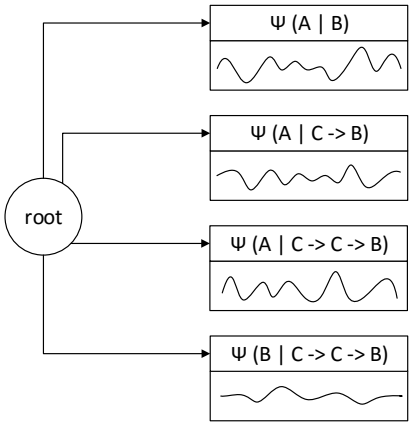

Figure 2: Example of flattened context tree with change of transition probabilities in time series.

Table 2: Flattened Context Tree vs. Regular Context Tree

	FCT*	RCT*
Node semantics	(context, target)	context
Tree depth	1	highest order
Probability time series	embedded	none

*FCT/RCT: flattened/regular context tree

2.2.2 Flattened Context Tree

We present *flattened context tree* and define the pruning strategy to facilitate distributed tree operations for our time-inhomogeneous Markov model. We prove that our parallel pruning strategy preserves the tree structure: one branch can be pruned iff all its children are pruned.

A flattened context tree T_F only has a depth of two: depth 0: root, and depth 1: all data nodes. Each depth-1 node $n_{\bar{p}}$ corresponds to a path $\bar{p} = (\bar{c}, t) = (s_{-y}, \ldots, s_{-2}, s_{-1}, s_0, t)$ and it records a time series of transition probability $\Psi(\tau|\bar{c}) = \{P_t(\tau|\bar{c}) : t \in T\}$. In comparison, a node $n_{\bar{c}}$ in T_C stores the distribution of transition probabilities according to context $\bar{c}$. Table 2 shows the most significant differences between our flattened context tree and regular context tree. We illustrate the structure of flattened context tree in Figure 2.

The advantage of the flattened tree structure is that each node can be processed independently of other nodes, which enables fine-grained parallel probability computing and pruning for each $(\bar{c}, \tau)$ pair. Furthermore, different nodes in T_F can be processed on a different processing unit in a distributed manner to scale out the process.

Pruning Pruning is performed for individual nodes in parallel. A node $n_{\bar{p}}$ is pruned away if $\bar{p}$ is rarely visited through a large segment of the monitored time period. The criteria can be measured as the total number of *path visits* or the total number of *path appearances* during the overall monitored time period. The latter yields *True* or *False* in each inspection window and sums the total number of *True* for the overall time period.

THEOREM 1. *In a flattened context tree T_F, a node $n_{\bar{p}}$ records the target transition probability time series of path $\bar{p}$. $\bar{p}'$ denotes a suffix string of $\bar{p}$. $n_{\bar{p}}$ in T_F can be pruned if $n_{\bar{p}'}$ (node corresponding to $\bar{p}'$) can be pruned.*

PROOF. If a path $\bar{p}$ is visited, its suffix path $\bar{p}'$ is visited. And two different paths $\bar{p_1}$ and $\bar{p_2}$ can share the same suffix path $\bar{p}'$. So $V(\bar{p}') \geq V(\bar{p})$ where $V(\bar{p})$ is the number of path $\bar{p}$ visits. Given $\mathcal{T}_{pv}$ as the pruning threshold for path visits, $V(\bar{p}) < \mathcal{T}_{pv}$ holds if $V(\bar{p}') < \mathcal{T}_{pv}$. □

If one restructures a flattened context tree T_F back to a regular context tree T_C, Theorem 1 guarantees that all children of a branch node are pruned away before the branch node is pruned. It is consistent with the standard pruning strategy presented in Section 2.2.1.

Time-variant capability Time series information is embedded into each node $n_{\bar{p}}$ in T_F, so T_F reflects the change of the corresponding Markov model through time. Change of E_M (discussed in Section 2.1) is distributed across $\Psi(\tau|\bar{c})$ in each node. Changes of V_M (discussed in Section 2.1) are also stored in new or obsolete nodes if not pruned.

2.3 Growing the Flattened Context Tree

We use a sliding window w to aggregate sessions through time and yield transition matrices of our time-inhomogeneous Markov model at different times. Time series yielded from nodes in the flattened context tree are extended when new sessions are consumed and the tree has grown.

2.3.1 Session Batch

Session batch is a set of sessions. It is the smallest sliding/stepping unit for w. Sessions are batched according to the timestamp of its first visited site. A session may last across several batch time periods, but the *entire* session is recorded only once in the first batch it appears[2]. The timestamp of the session batch is the start of the session batch.

Session batches do not interference with each other, and they can be preprocessed in parallel to facilitate the tree construction. In each session batch:

i) All paths at different lengths are identified (through n-gram with variable-n) and parsed into tuples $(\bar{c}, \tau)$.
ii) The counts of each tuple are accumulated.
iii) A set of 4-tuples $(\bar{c}, \tau, t_b, \eta_{(\bar{c},\tau)})$ is yielded as the session batch digest where t_b is the session batch timestamp and $\eta_{(\bar{c},\tau)}$ is the count of tuple $(\bar{c}, \tau)$ in the batch.

2.3.2 Sliding Window

The sliding window w covers a fixed number of session batches and each slide/step takes in a new session batch and abandons the earliest batch in the previous w.

In each sliding position, three operations are performed:

i) 4-tuples $(\bar{c}, \tau, t_w, \eta_{(\bar{c},\tau)})$ are accumulated from session batch digests where t_w is the timestamp of the earliest session batch in the window.
ii) 3-tuples $(\bar{c}, t_w, \eta_{\bar{c}})$ are accumulated from the 4-tuples where $\eta_{\bar{c}}$ is the count of context $\bar{c}$ in the window.
iii) A set of 4-tuples $(\bar{c}, \tau, t_w, P(\tau|\bar{c}))$ is yielded as the window digest where $P(\tau|\bar{c}) = \frac{\eta_{(\bar{c},\tau)}}{\eta_{\bar{c}}}$.

[2]Our current design does not support streaming because it requires the entire user session to finish before it can be sessionized and batched.

Table 3: Description of Spark Runtime Stages of our DECT Prototype

Functionality	#(SS)	Description
Input handling	1	reading user session data from HDFS
Modeling	19	n-gram generation, batching, in-window aggregation, probability calculation, pruning
Time series generation	9	assembling time series and storing them onto HDFS
Statistics generation	3	generating and yielding statistics throughout the entire processing procedure

#(SS): number of Spark runtime stages

2.3.3 Flattened Context Tree Update

Digests of w at different positions are aggregated according to tuple $(\bar{c}, \tau)$ in the window digest. Each tuple forms a depth-1 node $n_{\bar{p}}$ in the flattened context tree T_F. The time series at each node ($\Psi(\tau|\bar{c})$ at node $n_{\bar{p}}$ where $\bar{p} = (\bar{c}, \tau)$) is extended when a new window position is processed. w at different positions of a single node can be computed in parallel if all session batches are known.

Pruning of T_F can be performed at any time based on the generated time series at nodes. As discussed in Section 2.2.2, pruning is performed at nodes that are rarely visited. If $n_{\bar{p}}$ is pruned, $\Psi(\tau|\bar{c})$ prior to the pruning action is lost. $n_{\bar{p}}$ can be added back to T_F if it becomes popular in the future, but without the segment of $\Psi(\tau|\bar{c})$ when it is pruned away.

2.3.4 Time Series Production and its Applications

T_F records the user behavior pattern evolvement and it can be consumed by a variety of time series analysis tools (e.g., anomaly detection, path prediction).

User behavior evolvement data generation. The user behavior evolvement data are yielded in three major forms for post-processing and analytics:

- $\Psi(\tau|\bar{c})$: individual time series for each $(\bar{c}, \tau)$ pair
- $\{\Psi(\tau_i|\bar{c}) \mid 0 \leq i \leq \kappa_{\bar{c}}\}$: high-dimensional time series for all targets of a context $\bar{c}$ where $\kappa_{\bar{c}}$ is the total number of reachable targets of context $\bar{c}$.
- $\{\Psi(\tau|\bar{c}_i) \mid 0 \leq i \leq \varrho_\tau\}$: a collection of time series for a target τ where ϱ_τ is the total number of contexts which can reach the target τ.

$\{\Psi(\tau_i|\bar{c}) \mid 0 \leq i \leq \kappa_{\bar{c}}\}$ forms a high-dimensional time series where each dimension is $\Psi(\tau_i|\bar{c})$. We write $\{\Psi(\tau|\bar{c}_i) \mid 0 \leq i \leq \varrho_\tau\}$ as a collection because each $\Psi(\tau|\bar{c}')$ is not independent of $\Psi(\tau|\bar{c})$ where $\bar{c}'$ is a suffix of $\bar{c}$, yet it is quite useful to compare $\Psi(\tau|\bar{c})$ with $\Psi(\tau|\bar{c}')$.

User behavior evolvement data analysis. Three most important components of an aggregated user behavior time series are *trend*, *seasonality*, and *irregular component*.

Trend $\Psi_T(\tau|\bar{c})$ describes long-term movement without calendar related and irregular effects.

Seasonality $\Psi_S(\tau|\bar{c})$ characterizes regular cyclic movements influenced by seasonal factors.

Irregular component $\Psi_I(\tau|\bar{c})$ records non-systematic and unpredictable component(s) after trend and seasonal components are removed from the signal.

Many user behavior time series $\Psi(\tau|\bar{c})$ can be very well decomposed into the three components as described in (1).

$$\Psi(\tau|\bar{c}) = \Psi_T(\tau|\bar{c}) + \Psi_S(\tau|\bar{c}) + \Psi_I(\tau|\bar{c}) \quad (1)$$

$\Psi_S(\tau|\bar{c})$ can be further divided into daily/weekly seasonal components as found in our experiments.

Two major applications of our model are described next.

Anomaly Detection aims to discover anomalous user behavior with respect to specific visiting paths. A *spike* or a *ravine* in a time series could indicate breaking news, flash crowds, Denial-of-Service attacks, service failures, etc. A plateau appearing in the *trend* component of a time series may indicate a persistent attack or a test for a new feature. User behavior evolvement data $\Psi(\tau|\bar{c})$ and $\{\Psi(\tau_i|\bar{c}) \mid 0 \leq i \leq \kappa_{\bar{c}}\}$ are yielded for anomaly detection.

Ad Click Prediction is an application to predict how likely a user will click an ad on his/her current visiting site given his/her visiting path of the current session. Different visiting paths leading to the same site may give different ad click rates, and the probability trends of different paths may be different. User behavior evolvement data $\Psi(\tau|\bar{c})$ and $\{\Psi(\tau|\bar{c}_i) \mid 0 \leq i \leq \varrho_\tau\}$ are yielded for ad click prediction.

3. IMPLEMENTATION

We implement DECT via Apache Spark using Scala. We deploy DECT on top of Yahoo! infrastructure to support anomaly detection and other services on Yahoo! network.

Our DECT implementation is open-sourced on github [19]. The implementation takes advantage of scalable and robust transformations on resilient distributed dataset (RDD) in Spark, e.g., `mapValues` and `join`. DECT compiles to 32 Spark stages at JVM runtime (shown in Table 3).

Our implementation consumes plaintext session data stored on HDFS where each line records a user session[3]. A user session consists of a timestamp t_s and a sequence of visited sites $E = \{s_0, s_1, \dots\}$. DECT digests the plaintext session data and yields two types of information: *i)* time series harvested from the flattened context tree, stored on HDFS, and *ii)* statistics on processed data, e.g., total number of ith-order time series, printed to Spark log.

Our realization is optimized from the following aspects:

1. Session and path (n-gram) data are aggregated at early stages to minimize unnecessary duplicate data processing. For example, before generating and counting n-grams in each session E, same E with the same session batch timestamp t_b are counted and deduplicated.
2. Compact data structures are used to reduce storage and transmitting overhead, e.g., a context as a single JVM string, instead of an array of sites (JVM strings).
3. Job parameters are broadcasted, e.g., string splitter.
4. Partitioning strategies are manually specified to reduce data movement among worker nodes.

[3]We employ a `Pig` script to retrieve, sessionize and store raw user event data from HCatalog onto HDFS prior to DECT.

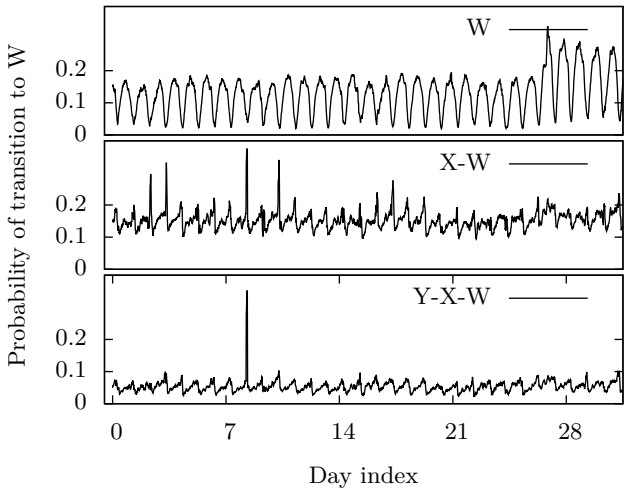

Figure 3: Higher-order path time series anomaly detection[5].

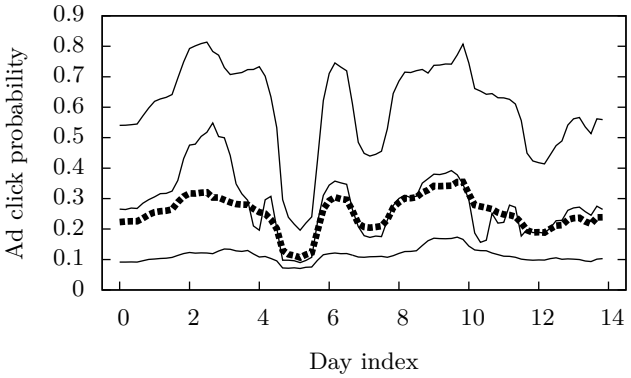

Figure 4: Ad click probabilities given different paths. The bold dotted line denotes the overall ad click rate of users on a site. Each thin line denotes ad click rates of users on this site coming from one specific visiting path[5].

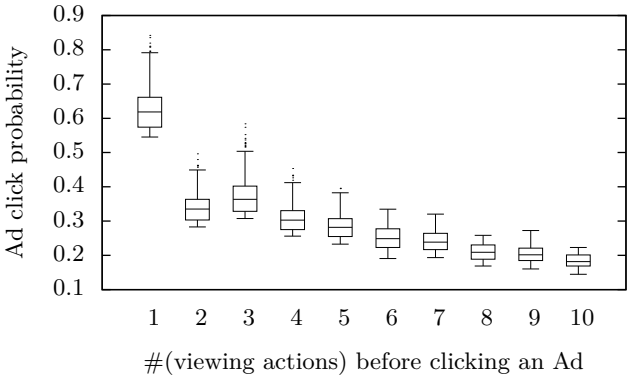

Figure 5: Ad click probability of a wanderlust[5].

4. EVALUATION

We conduct experiments on two Yahoo! daily user session datasets to answer the following key questions:

1. What do we benefit from our time-variant user behavior model over existing static stochastic models?
2. Is our design scalable to handle enterprise-wide tasks consisting of billions of sessions?

We evaluate DECT with all data collected through Yahoo! data highway. We analyze Yahoo! user activities within the first half of 2015. In the evaluation, we focus on user sessions within a single product, e.g., Yahoo! mail. Visits to alien sites during sessions are ignored[4].

We process user session data of Yahoo! US websites (English version) within two products separately: Yahoo! mail and Yahoo! finance[5]. Each site in a session is roughly a view in the model-view-controller (MVC) web architecture, and it has a unique URL.

4.1 Case Study: Anomaly Detection

Time series of site visits are commonly used as anomaly detection signals. However, if no context information is specified, anomaly signals of specific visiting paths are masked out by other signals. Therefore, it causes false negatives.

We pick a typical Yahoo! mail site W (i.e., τ in Section 2)[5] and show that anomalies in higher-order path signals are significant and can be revealed by DECT. We use DECT to compute visiting probabilities of W for all contexts $\{\bar{c}\}$ that exist. We then fed time series $\{\Psi(\tau_i|\bar{c}) \mid 0 \leq i \leq \kappa_{\bar{c}}\}$ yielded by DECT into EGADS for anomaly detection. EGADS is a generic and scalable framework for automated anomaly detection on large scale time-series data [10]. The entire anomaly detection application consists of DECT (time-variant user behavior modeling) and EGADS (time series anomaly detection).

Fig. 3 shows two higher-order anomalous time series identified by EGADS. The upper subfigure, a common seasonal time series across a month, is the visiting probability of W across all Yahoo! mail sites. One anomalous time series (site X to W) is detected during that month (several spikes in the middle subfigure). Another anomaly is found on the 8th day of visiting path "Y to X to W" in the lower subfigure. Any anomaly (spike) with respect to a higher-order path may be hidden in the time series of its suffix path.

4.2 Case Study: Ad Click Prediction

Existing ad click prediction techniques do not take historically visited paths into account. We run DECT on the Yahoo! finance dataset to show that such information is useful in distinguishing probabilities of ad clicks.

We draw the overall ad click rate on a Yahoo! finance site[5] in Fig. 4 with the bold dotted line. We then use DECT to investigate three ad click rate time series, each of which has a site previously visited (one-time context) before the target site. Fig. 4 shows that the click rate of users coming from one site can be 5 times higher than that of another.

Besides the finding that *ad click rates are related to user*

[4]DECT can be deployed at the client/browser side to model and analyze Internet-wide user behavior.

[5]According to Yahoo! data privacy requirements, *i)* detailed data statistics are not provided; *ii)* probabilities in figures are disguised while their relative positions are preserved.

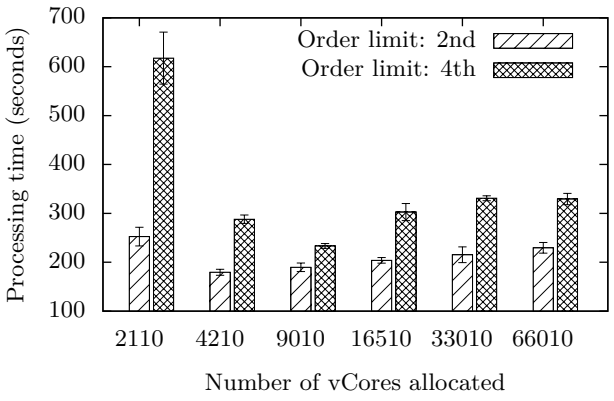

Figure 6: Performance pivot discovery of DECT.

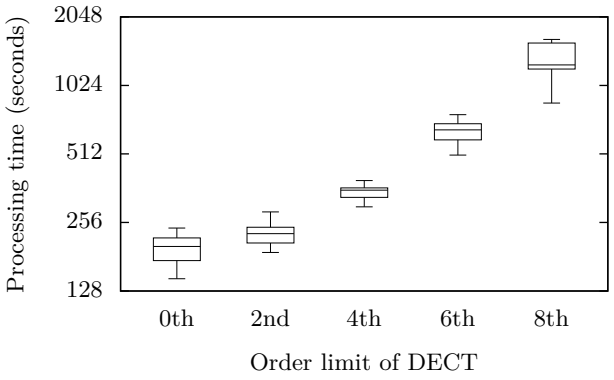

Figure 7: Order impact on overall computation complexity.

visiting paths, another interesting conclusion we reached is that *the more a user views articles/pages on a site, the less likely she will click an ad on that site.* We illustrate the decrease of ad click rates on a Yahoo! finance site[5] in Fig. 5. We explain the phenomenon that frequent readers tend to continuously consume target information, e.g., stock values, and ignore ads. Ads could be less effective and more annoying to frequent readers than normal visitors. This work is being deployed at Yahoo! for better ad-targeting.

4.3 Performance analysis

We demonstrate the performance of our implementation by processing a subset of Yahoo! mail data (around 0.1 billion user sessions, 10GB storage size on HDFS)[5].

Scalability and Performance Pivot. Our realization of DECT is based on the scalable Apache Spark framework. A distributed system results in an increasing amount of communicating/scheduling overhead when scaling out. We are interested in discovering performance pivots and parameter tuning on real-world datasets.

We conduct several groups of experiments with DECT running on different numbers of worker nodes. We measure the degree of parallelism via the maximum number of processing units (vCore)[6] concurrently allocated at any execution stage. Fig. 6 shows that our implementation scales out well before reaching a performance pivot. Performance pivots are reached for DECT with order limit 4 at 9010 vCores and order limit 2 at 4210 vCores. Increasing the number of processing units after the pivots wastes more in overhead than gaining better performance. The more complex the computation is, e.g., higher order limit, the larger amount of processing units are required to reach the pivot.

[6]The number of workers is linear to the number of vCores.

Magnitude of frequently visited higher-order paths. DECT is a variable-order Markov model. It employs a pruning procedure to remove time series of rarely visited higher-order paths. We are interested in the order impact on the overall computational complexity.

We execute DECT with various order limits. Because the total number of possible paths is exponential in path order, the results in Fig. 7 show that: *i)* the processing time increases exponentially with the increase of the order limit, and *ii)* the pruning procedure reduces the number of time series *by a constant factor* on Yahoo! mail dataset.

5. RELATED WORK

Our work is motivated by time-homogeneous Markov user behavior modeling, time series analysis, and evolutionary network analysis.

Time-homogeneous Markov modeling. Web user behavior has been studied for various purposes, such as PageRank [13], link prediction [17], document prefetching [23]. A variety of time-homogeneous Markov models have been tested to describe Internet user behavior [5]. The time-homogeneous indicates that the transition matrix of the Markov model does not change through time. We list some existing models classified by their Markov orders below.

- First-order Markov model: [12, 13]
- Second-order Markov model: [23]
- Higher-order Markov model: [15]
- Variable-order Markov model: [2, 5, 6]

Variable-order Markov models compute different orders for different paths to reduce storage expenses. The idea was proposed by Bühlmann and Wyner [3]. There exist two generic approaches to construct variable-order models.

Pruning-based approach: starting with a complete higher-order model and iteratively pruning low-entropy branches to get a incomplete tree, e.g., [6].

Growing-based approach: starting with a first-order Markov model and expanding leaves with inconsistent distribution into branches, e.g., [2].

Our design follows the former approach for straightforward parallel design. The operations of growing higher-order paths, i.e., slicing and clustering, are computational heavy and the results cannot be efficiently reused over time.

Time series analysis. A time series denotes the change of a variable over time [8]. Time series analysis has been applied to many fields including signal forecasting [9], data feature extraction [7] and anomaly detection [4, 10, 11, 20].

Time series analysis is widely used to detect anomalous user events in the industry. However, studied variables in existing systems are mostly primitive, e.g., counts of site visits. They only represent zeroth-order or first-order (one-hop) paths [10]. The prediction is fast but loses rich context information. DECT, in contrast, utilizes historical site visit-

ing information to provide more detailed signals for anomaly detection and ad click rate prediction as shown in Section 4.

Evolutionary network analysis. Dynamic networks appear in social networks, wireless sensor networks, Internet of Things, and the Web. The analysis of evolving networks provides a comprehensive understanding of such networks [1] and enables applications such as link prediction [18] and anomaly detection [16].

Graphs are generic models for dynamic network representation [1]. More specifically, dynamic networks usually generate complex cyclic graphs, and evolutionary network analysis heavily relies on unique properties of such graphs, e.g., community discovery [14]. Compared to cyclic graphs, variable-order Markov models are tree-equivalent structures. In our model, we bring some concepts from evolutionary network analysis, e.g., *change of* V_M and *change of* E_M. But, in general, it is currently unclear how evolutionary network analysis methods can be applied to dynamic web user behavior modeling.

6. CONCLUSIONS AND FUTURE WORK

This paper presents DECT, a scalable time-variant web user behavior model. It characterizes the changing nature of Internet user behavior with a variable-order time-inhomogeneous Markov model. DECT can be efficiently realized on scalable distributed frameworks, e.g., Apache Spark, to process large volumes of user behavior data. DECT enables time series analysis on individual or related sets of long (higher-order) user paths. We open-sourced DECT and deployed it at Yahoo! to support path time series analysis such as anomaly detection, click probability prediction and path trend discovery. In the future work, we plan to work on streaming pruning strategies to enable streaming user behavior processing using DECT.

7. REFERENCES

[1] C. Aggarwal and K. Subbian. Evolutionary network analysis: A survey. *ACM Computer Surveys*, 47(1):10:1–10:36, May 2014.

[2] J. Borges and M. Levene. Evaluating variable-length Markov chain models for analysis of user web navigation sessions. *IEEE Transaction on Knowledge and Data Engineering*, 19(4):441–452, April 2007.

[3] P. Bühlmann and A. J. Wyner. Variable length Markov chains. *The Annals of Statistics*, 27(2):480–513, April 1999.

[4] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computer Surveys*, 41(3):1–58, July 2009.

[5] F. Chierichetti, R. Kumar, P. Raghavan, and T. Sarlos. Are web users really Markovian? In *Proceedings of World Wide Web Conference*, pages 609–618, New York, NY, USA, 2012.

[6] M. Deshpande and G. Karypis. Selective Markov models for predicting web page accesses. *ACM Transactions on Internet Technology*, 4(2):163–184, May 2004.

[7] P. Esling and C. Agon. Time-series data mining. *ACM Computing Surveys*, 45(1):12, 2012.

[8] J. D. Hamilton. *Time series analysis*, volume 2. Princeton university press Princeton, 1994.

[9] R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.

[10] N. Laptev, S. Amizadeh, and I. Flint. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1939–1947, 2015.

[11] N. Laptev, R. Hyndman, and E. Wang. Large-scale unusual time series detection. In *Proceedings of IEEE International Conference on Data Mining*, 2015.

[12] Z. Li and J. Tian. Testing the suitability of Markov chains as web usage models. In *Proceedings of Annual International Computers Software and Applications Conference*, pages 356–361, November 2003.

[13] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[14] S. Parthasarathy, Y. Ruan, and V. Satuluri. Community discovery in social networks: Applications, methods and emerging trends. In C. C. Aggarwal, editor, *Social Network Data Analytics*, pages 79–113. Springer US, 2011.

[15] P. Pirolli and J. Pitkow. Distributions of surfers' paths through the World Wide Web: Empirical characterizations. *World Wide Web*, 2(1-2):29–45, 1999.

[16] S. Ranshous, S. Shen, D. Koutra, S. Harenberg, C. Faloutsos, and N. F. Samatova. Anomaly detection in dynamic networks: a survey. *Wiley Interdisciplinary Reviews: Computational Statistics*, 7(3):223–247, 2015.

[17] R. R. Sarukkai. Link prediction and path analysis using Markov chains. *Computer Networks*, 33(1):377–386, 2000.

[18] R. R. Sarukkai. Link prediction and path analysis using Markov chains. *Computer Networks*, 33(1–6):377–386, 2000.

[19] X. Shu. Distributed evolving context tree (DECT), https://github.com/subbyte/DECT.

[20] O. Vallis, J. Hochenbaum, and A. Kejariwal. A novel technique for long-term anomaly detection in the cloud. In *Proceedings of USENIX HotCloud Workshop*, pages 15–15, Philadelphia, PA, June 2014.

[21] Y. Xie and S. zheng Yu. Monitoring the application-layer DDoS attacks for popular websites. *IEEE/ACM Transactions on Networking*, 17(1):15–25, Feb 2009.

[22] S. Yu, G. Zhao, S. Guo, Y. Xiang, and A. Vasilakos. Browsing behavior mimicking attacks on popular web sites for large botnets. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 947–951, April 2011.

[23] I. Zukerman, D. W. Albrecht, and A. E. Nicholson. Predicting users' requests on the WWW. In *Proceedings of the 7th International Conference on User Modeling*, pages 275–284, Secaucus, NJ, USA, 1999. Springer.