Community Systems Research at Yahoo!

Community Systems Group
Yahoo! Research
Sunnyvale, CA 94089 and New York, NY 10018
{ramakris}@yahoo-inc.com

ABSTRACT

The web and its continued evolution present unprecedented opportunities for database researchers and practitioners to deliver unique user experiences that are not possible traditionally, e.g., mass collaborations through (automatically) established online communities and exploration of large scale structured information. Along with these opportunities, however, come significant challenges. The challenges are twofold: *systems*, the infrastructures that allow us to deliver information at scale; and *community*, the applications that deliver the next generation of web experiences centered around people and social networks. In this paper, we describe the ongoing research efforts within the *Community Systems* group here at Yahoo! Research to address these challenges.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Systems and Software

General Terms

Design, Management, Performance

Keywords

web-scale system, web community, web data management

1. INTRODUCTION AND OVERVIEW

In mid-2005, Yahoo! Inc. began an ambitious program to create a world-class industrial research lab focusing on how to deliver services over the web to a range of stakeholders, including advertisers, site owners, content publishers, and users. Yahoo! Research now has groups in the following areas: Community Systems, Computational Advertising, Machine Learning, Media Experience and Design, Microeconomics, and Web Search. In this paper, we present an overview of the **Community Systems** group, which now includes Vipul Agarwal, Sihem Amer-Yahia, Philip Bohannon, Brian Cooper, Nilesh Dalvi, Minos Garofalakis, Arun Iyer, Vinay Kakade, Daniel Kifer, Raghu Ramakrishnan, Adam Silberstein, Utkarsh Srivastava, Ramana Yerneni, and Cong Yu. We also collaborate closely with others from Yahoo! Research, including Marcus Fontoura, Vanja Josifovski, Ravi Kumar, Cameron Marlow, Srujana Merugu, Chris Olston, Bo Pang, Seung-Taek Park, Benjamin Reed, Sathiya Keerthi Selvaraj, Jayavel Shanmugasundaram, Andrew Tomkins, Sergei Vassilvitskii, and Erik Vee, and from other parts of Yahoo!.

We believe that the web has introduced significant new challenges and opportunities [14, 13] at many levels—how we deliver applications to customers, how we monetize those applications, how we build and support those applications, and even the very nature of the applications that are made possible by leveraging the information accessible through the web. The goal of our group is to take on these challenges, in particular those that are central to building, supporting, and analyzing applications involving large user communities, and to enable Yahoo! to capitalize on the opportunities presented on the web. In this short paper, we present an overview of some of the projects we are undertaking, organized into sections that reflect the goals of the group.

2. WEB-SCALE INFRASTRUCTURE

In this section, we describe the two main systems we are currently building for managing web-scale data: *PNUTS* and *Pig*. We also briefly discuss *AppForge*, a system that enables GUI-driven development of hosted web applications.

2.1 PNUTS

The PNUTS project[1] is to build a massively scalable data management service to provide back-end support for Yahoo!'s web workloads. At Yahoo! scale, massive parallelism and distribution are necessary to provide acceptable latency and high throughput for web workloads. In particular, social and community applications significantly increase data needs by injecting huge amounts of user generated content, and by requiring complex relationships among a large number of users to be efficiently maintained and queried. PNUTS partitions and replicates data over thousands or tens of thousands of servers in order to handle data at this scale, while providing clean abstractions that make it easier for applications to deal with this complexity.

Four guiding principles shape the design of PNUTS. First, it provides *high performance* at large scale by using asynchrony, weak consistency and loose coupling. Second, it uses automated replication and failure recovery to ensure *high availability*. Third, it is designed to be *easy to use, operate and scale*. Ease of use means that the external abstractions hide much of the complexity of the underlying distributed, replicated system. Ease of operation implies extensive self-management and self-tuning. Ease of scaling means that adding capacity is as simple as plugging in new machines and turning them on. Fourth, PNUTS provides multiple rich access methods, including multiple types of primary tables and secondary indexing.

PNUTS is both a research project and a key piece of Yahoo!'s next generation platform architecture. The system is being designed and built as a collaboration between Yahoo! Research and Yahoo!'s Platform Engineering group,

[1]PNUTS is an acronym that reflects a jar of peanuts we were snacking on during the project kickoff; the current expansion is "Platform for Nimble Universal Table Storage."

and some initial components of the system have already entered production use.

System Architecture: PNUTS is a centrally hosted and managed data service. Multiple applications concurrently connect to the service to store and query data. This shared service model addresses one of the main data problems faced by Yahoo! today: applications often have to set up, maintain and scale their own data services, a significant drain on business resources and an impediment to the development of new features.

The physical data in a PNUTS table is horizontally partitioned over a large number of storage servers. The assignment of data partitions to storage servers is flexible, and partitions can be re-shuffled to balance load or recover from storage server failures. PNUTS tables can be hash tables or ordered tables; hash tables provide fast single record lookup while ordered tables are optimized for fast range scans. Storage servers are also responsible for evaluating queries, collecting and delivering query results, ensuring updates are applied consistently, updating secondary indexes, and so on.

Each table is replicated to multiple geographic regions to provide both disaster recovery and low-latency access for international users. Replication between regions is asynchronous, utilizing a topic-based pub/sub system. A query routing layer isolates applications from having to know the current location of a given table partition. Developers declare the data types of table columns, ensuring that predicates are typed properly and that data is sorted correctly (in the case of ordered tables). New attributes can be cheaply added to existing schemas; a catalog update is required but existing stored data is not modified. PNUTS currently does not support other kinds of schema constraints, such as foreign keys, because of the cost to enforce such constraints over massive, partitioned data sets.

Ongoing Research: Hosted data management services are a new and significant direction for database research. It is the key technology behind the software-as-a-service (SaaS) paradigm, which is rapidly gaining in popularity, since most (if not all) hosted applications need to manage large-scale application data. Developing hosted data services is fundamentally different from developing shrink-wrapped softwares deployed by each customer organization; in effect, we are designing systems that allow us to serve as "DBA to the world," and this raises a slew of challenges, including scaling, robustness, support for applications designed to be rented by multiple "tenants," differential quality of service guarantees to different tenants, flexible access controls, and self-tuning for virtually every aspect of the system.

The basic architecture of the system has been designed and many components have been implemented, but the PNUTS project is still at an early stage, and there are several active, ongoing research thrusts; we outline some of these below:

The first research thrust is to develop a stronger consistency model. Existing mechanisms for providing ACID transactions for distributed databases do not perform well at our scale. Our basic consistency model guarantees all readers see a consistent history for individual records, but does not provide guarantees for reads or writes that span multiple records. In our model, applications can read and then write a record, specifying that the write succeeds only if the record has not changed since the previous read. However, we are currently examining ways to use the consistency primitives in this basic model to build more complex transactions, where multiple records can be read or written in a way that is serializable with respect to other transactions.

A second active area of research is maximizing the efficiency of bulk operations. Although the system is designed to provide high throughput for lots of parallel queries, a sudden burst of load caused by a bulk insert or bulk read can still overwhelm the system, especially if it induces a hotspot. To deal with these issues, we are developing mechanisms by which the system accepts a bulk request, and then reshuffles the individual requests to ensure efficient resource usage and maximum parallelism. Thus, a bulk read of many records will be reshuffled into a bunch of parallel reads that go to different storage servers, while a bulk insert requires reshuffling the inserted data to ensure maximum parallelism.

A third area of research involves guaranteeing quality of service. Because PNUTS is a hosted, shared platform, many applications will be using the system concurrently. The challenge is to ensure that every application receives a fair proportion of the resources, while ensuring that load spikes can be absorbed by otherwise idle capacity. Our approach is to extend traditional ideas in admission control, rate limiting and pre-emption (e.g., queries might get rejected, might get processed at a deliberately slow pace, or might get interrupted) to work in a large scale, distributed database.

2.2 Pig

The *Pig* project addresses the problem of ad-hoc analysis of extremely large data sets, by users whose primary role is software development. This scenario arises in internet companies, where software services are routinely deployed and refined based on analyzing the recorded behavior of users. It differs from those for which SQL and traditional database systems were designed, and hence dictate the need for a new data analysis language and a new underlying system. The key differentiating factors are:

Programmers as users: The data analysts are typically experienced programmers who tend to think procedurally and often find the declarative style of the SQL language to be overly restrictive.

Custom processing: A large part of such data analysis consists of custom, domain-specific processing that is difficult to express in SQL and requires extensive use of user-defined functions.

Scale: The data scale is well beyond the capacity of single-node databases, and parallel database solutions (e.g., Teradata) can be prohibitively expensive to purchase and operate, due in part to specialized hardware. Ideally, a solution that can utilize cheap commodity hardware is desired.

These considerations have led us to design a new system for data analysis called *Pig*, along with a new data analysis language called *Pig Latin*. Pig Latin has a procedural flavor like a programming language, and has extensive support for user-defined functions. At the same time, Pig Latin retains SQL-like high-level constructs such as filtering, joining, aggregation, and grouping. The use of such high-level constructs allows for optimization and parallelization of queries, which is in contrast to *Map-Reduce* [5] where all computation must be structured as opaque map or reduce functions. This not only makes simple operations such as projecting and filtering cumbersome to write, but also impedes optimizations such as filter re-ordering or multi-query sharing.

To give a flavor of Pig Latin, we present a simple example. Consider two data sets associated with a search engine:

`queryResults(queryString, url, rank)`, and `urlInfo(url, pageRank)`. The `queryResults` table records the results of search engine queries (which URLs were displayed at what rank positions); the `urlInfo` table gives the precomputed PageRanks for each URL. Suppose a user wishes to identify search query strings for which the top PageRank page did not occur among the top five results. The user writes a simple sequential program called `checkTop5` that, given the set of results for a `queryString`, determines whether PageRank played a dominant role. The following simple Pig Latin program performs the overall analysis:

```
a = JOIN queryResults BY url, urlInfo BY url;
b = GROUP a BY queryString;
c = FILTER b BY checkTop5(*);
```

A Pig Latin program consists of a sequence of assignments to table variables, (e.g., `a`, `b`, `c`). The right-hand side of each assignment expresses a data transformation step such as join or filter. The available data transformation primitives roughly correspond to the relational algebra operators, with extensions to accommodate aggregation (not discussed here) and user-defined functions (discussed shortly). This algebra-style querying resembles conventional programming (e.g., Java or Python) more closely than does SQL, and for that reason the programmers with whom we work at Yahoo! tend to find it easier to use than SQL.

Another aspect of Pig Latin that adds significant appeal and power for our programmer user base is the ability to incorporate user-defined functions easily into essentially any operation, including filter (as illustrated in the above example), group-by, join and (of course) aggregate. Typically these user-defined functions process small amounts of data at a time, so there is no need to parallelize a single invocation. Hence the opaqueness of user-defined functions does not present a performance problem. (For functions that are invoked over large data segments, we offer an API for encoding them as distributive or algebraic functions that are amenable to parallelization.) Basic large-scale operations such as grouping are exposed through high-level primitives, permitting parallel evaluation.

The full details of Pig Latin can be found on our website [2]. Pig Latin is fully implemented (currently, by compilation into *Hadoop* [1], an open-source implementation of Map-Reduce) and is being used within Yahoo! for data analysis. There are two research directions that we are currently pursuing:

Pig Body (A native query processing engine for Pig): As stated above, our initial implementation of Pig compiles Pig Latin programs into Map-Reduce jobs. We are working on a next-generation implementation whose query processing engine more closely resembles traditional parallel DBMSs such as Gamma [7], although on a much larger scale. Once our native query processing engine is in place, we intend to explore opportunities for aggressive sharing of work across independently submitted queries. The potential advantage of sharing work is large, because queries tend to run for hours or days, and often exhibit significant overlap in the data they access and the processing they perform. For example, Yahoo! programmers circulate via email a command sequence for scanning the web crawl while filtering out spam and foreign-language pages, which is issued regularly as a prefix to various custom data analysis tasks. To exploit such cross-query commonalities, we will implement view materialization and multiquery execution techniques, and optimization algorithms to govern these techniques. Extensive use of user-defined functions, along with the chaotic nature of our cluster computing environment, make conventional model-based approaches to the associated optimization problems dubious, so we intend to explore adaptive approaches instead.

iPig (Incremental evaluation in Pig): Often, users want to run queries over continuously updated data such as query logs or web crawls. Currently, the query has to be reissued when the data has changed, and results in a recomputation from scratch. The iPig effort seeks to solve this problem by allowing users to register continuous queries whose answers will be incrementally updated as and when data updates arrive. One of the main goals of iPig is to be fully compatible with Pig, so that Pig programs (and user-defined functions) can be written while being fully agnostic to the incremental recomputation. Providing such a simple programming model, while still maintaining efficiency is one of the main challenges. Another main challenge is managing the state associated with incremental computation of user-defined functions in an efficient and fault-tolerant manner on top of a cluster of failure-prone machines.

2.3 AppForge: Graphical Development of Hosted Web Applications

AppForge is a graphical application development tool for "power-users" or developers who wish to develop community applications, but who do not have any prior programming expertise or database knowledge. Based on the WYSWYG (What You See is What You Get) paradigm, it allows power-users to create an entire application, including the underlying database and application logic, simply by creating the application screens that will be seen by end-users. Equally important, AppForge also hosts the application, thereby relieving the power-user of the burden of deploying and maintaining applications.

One possible application of AppForge is in the context of online communities such as Yahoo! Groups. As an illustration, consider a Yahoo! Group that is devoted to bicycle club fans. The Yahoo! group provides them with a message board for sharing messages. However, it does not provide any advanced functionality that is specific to a particular group. (Since Yahoo! Groups has a wide variety of groups, ranging from book clubs to bicycle clubs to child care groups, supporting all group-specific functionalities is beyond its capacity.) The bicycle club members may wish to create an application that allows members to car-pool for bicycle rides based on how many bicycle racks are available on a given car, the location where people live, etc. Today, they can only develop this application by explicitly programming and hosting it, which is usually beyond the skill and knowledge of the club members. With AppForge, the group moderator can graphically create an application tailored to the particular group, without having to know anything about programming, database management, or application deployment.

There are three technical components of AppForge. The first component is the Schema and Application Logic Inference Module, which infers the underlying relational schema and application logic based on a set of WYS-WYG graphical primitives that are exposed to the user. The second component is a Schema Navigation Module, which enables users to navigate entities and relationships using menus, without

having to understand the formal Entity-Relational model. The third component is the Hosted Application Module that deals with issues related to application hosting.

Recently, database usability [10] has received significant attention within the database community, we believe AppForge addresses many of the similar usability challenges arising from dealing with complex databases.

3. DISCOVERING STRUCTURE

In this section, we describe two main research projects, Purple SOX and GUESTS, that aim at discovering and exploring structured information on the web.

3.1 Purple SOX

Existing information extraction systems, as a rule, require careful design and tuning by engineers before achieving acceptable quality on a particular domain such as shopping, product reviews, or bibliographies. The design phase typically involves analyzing the extraction domain and formulating an approach in the form of an "extraction pipeline," which is then populated with tools for tasks such as page classification, word sequence labeling, etc. The performance of these tools needs to be evaluated and extensively tuned via programming, feature selection, retraining, etc. The whole process is time-consuming and expensive, and the result is generally a good quality, but highly *domain-specific* system. This domain specificity is in stark contrast to the ultimate goal of *domain scalability*, i.e., the ability to apply information extraction to a wide variety of domains at reasonable cost.

In response, the Purple SOX project seeks to substantially decrease the cost of developing information extraction systems with acceptable quality for a large number of domains. The project proceeds along two key directions: (a) the creation of an Extraction Management System (EMS), and (b) the development of flexible and transferable Extraction Operator Library (EOL). Purple SOX has grown out of the information extraction component of Cimple [6], a joint community information management project with the University of Wisconsin, and is closely integrated with the Vertex information extraction platform, an existing internal platform at Yahoo!. Purple SOX is also influenced by information-extraction platforms such as Avatar [11], although it differs in its architecture and web-facing emphasis from Avatar.

Extraction Management: The design of the Purple SOX EMS is motivated by the need for a system that is extensible, explainable, autonomous and social. We briefly describe each of these principles and then outline the architecture of the system. First and foremost, an information extraction system that seeks to apply to a large number of domains cannot limit itself to a small number of extraction components or operators. In order to be *extensible* to a wide variety of extraction technologies, it should be possible to add new *extraction operators* to the system in a straightforward, declarative manner. If one accepts the loose analogy of information extraction as a "query" over the extraction corpus, it might seem straightforward to model each extraction technique as an "external function" of a traditional query processing or information gathering model. While following this approach at a high level, we find that a number of subtle challenges arise due to issues such as uncertainty and the role of training and feature selection.

The second principle is that extraction should be *explainable*; that is, the results of extraction efforts must be available in a form that supports browsing and analysis—what is working, and what is not? This in turn requires that partial extraction results be recorded in a data management system in which the history of events can be traced through a lineage tracking mechanism. Since the extraction results are uncertain, this uncertainty must also be tracked to avoid showing low quality data to users.

The third principle, *autonomy*, requires that extraction tasks be carried out and improved with minimal active management. For example, the system must be able to evolve pipelines by substituting different applicable technologies in an effort to improve quality. To support planning, the system must understand the operator capabilities in a semantic as well as syntactic way. Further, the quality of each extracted datum must be automatically estimated based on available evidence across a variety of sources, extraction algorithms, and human input.

Finally, we do not believe that freedom from human input is possible, and instead a clear goal is to replace expert tuning with extensive *social input*, including positive examples, occasional markup, and a variety of feedback on the quality of extraction results.

The architecture of the Purple SOX EMS is actively evolving to meet the above design requirements, and current high level components include: 1) a *probabilistic data model* supporting highly flexible typing, easy extension with new types, and tracking of confidence and lineage on a per-attribute basis, 2) a *declarative operator framework* for information extraction components including specification of lineage and optional confidence on extraction outputs, 3) a *reconciler* charged with aggregating a variety of opinions concerning information in the data model to determine a "system confidence," and 4) a *planner* to select among alternative approaches for a given extraction task. Numerous research challenges arise in the design and validation of these components including modeling and estimation of uncertainty across wrapped operators, new challenges in extraction planning, the need to extend simple models of external functions to handle trainable operators, factoring work out of repeated extraction, etc. An equal number of system challenges involving performance (especially of lineage data) and parallelization are expected to arise.

Extraction Techniques: Purple SOX EOL is a suite of machine learning and rule-based techniques necessary for building structured community portals. There are multiple objectives in creating this operator library. The first and foremost one is to accumulate and create tools that can support key extraction tasks such as entity discovery and disambiguation, record extraction and general relationship discovery. An important distinguishing feature of the Purple SOX EOL is the fact that each of these extraction operators is accompanied by a description of its input/output and the associated dependencies in terms of the semantics of the data model, which in turn induces a natural hierarchy over the operators. A secondary objective is to test the expandability features of Purple SOX EMS by wrapping existing machine learning tool boxes such as Weka, Elefant. Designing an infrastructure that allows specification of preprocessing (e.g., feature extraction/learning rules) and evaluation steps in a declarative fashion is another important aspect. Last, but not the least, a key goal of Purple SOX is to develop new learning methodologies for structured predic-

tion that address challenges arising in domain adaptation, learning from partial user feedback/constraints, and utility based learning.

3.2 GUESTS

An ever-growing number of users participate in social content sites such as Flickr, del.icio.us, and YouTube, making friends and sharing content. Users come to these sites to find out about general trends (e.g., the most popular tags or the most recently tagged URLs), as well as look for more specific information (e.g., the recent posts of their friends). The ability to help users *sift* through the large amount of content on social sites is a challenging question which requires combining techniques from databases, information retrieval and machine learning [3]. Leveraging the users' social behavior to recommend new content is a key technical challenge in social content sites. While explicitly declared social ties (friends and family) are known to users, implicit ones induced by common social behaviors (e.g., tagging in del.icio.us) are a greater indicator of shared interest and should be leveraged in recommending new content [12].

Challenges. GUESTS (Groups of UsErs going Social in web Two.0 Search) aims to leverage explicit and implicit social ties to *guide users in the personalized discovery of new content* and *involve them in the process.* Using del.icio.us as an example, the key challenges are:

Network discovery: In order to help users discover new social ties that are relevant, we need to define techniques for deriving social networks from common interests. Moreover, the ability to *explain* derived social networks is crucial to guide users in their discovery process. We propose to associate an *interest topic* to each derived network (e.g., the user can see the network of people who share common interest with him in Cooking or the network of people who share his interest in French Poetry). We also propose to explain each *social tie* in a network by *a list of tags* which are common to the two users forming the tie. Users should also be able to select social ties for further processing (e.g., add a person to my active network).

Vocabulary discovery: Users should have the ability to request a *network-specific vocabulary* as a set of topics of interest to members of that network. For example, the *most popular tags* unique to a given network will reveal more specific information. More precisely, a social network may be using the term "menu" to mean "restaurant menu" while another one to mean "software menu". A user can belong to both networks. Vocabularies are used as a mean to *explain* the social network.

Content discovery: The ability to recommend and explain new content using social networks and their associated vocabularies is at the core of the system. Users want to select a social network and view recommended content which are identified based on their popularity among members of the selected network. The ability to issue a search query and see query-relevant items which are most relevant toward specific networks is another important task. Finally, the system should explain recommended items (e.g., this is a health and nutrition site and is popular among members of your Cooking network).

We now discuss how the above challenges are implemented in GUESTS. Our implementation uses del.icio.us datasets (bookmarked URLs are referred to as items).

Tag Analysis. We implemented a tag analysis tool which is based on *co-occurrence of tags* (akin to frequent itemset mining) in order to synthesize tags into *topics.* This analysis can be done over any set of tag pairs (e.g., those used by members of a selected social network). We use the technique described in [15], where we look for pairs of tags where one is subsumed by the other based on co-occurrence. For a tag t, we denote by $I(t)$ the set of all items that were tagged with t. Given tags t_1 and t_2, we denote t_2 as the topic of t_1, i.e., $t_1 \Rightarrow t_2$ iff $|I(t_1) \cap I(t_2)|/|I(t_1)| \geq$ threshold and $|I(t_1) \cap I(t_2)|/|I(t_2)| <$ threshold.

The analysis may detect that whenever the tag "j2ee" is used, the tag "java" is also used (with a given confidence threshold), but not vice versa, and will therefore denote "java" as a topic. When both tags subsume each other, we consider them synonyms and part of the same topic. Other interesting connections include "chowhound" (a restaurant critic site) being associated with "food", and "fundraising" being associated with "politics" in one social network and with "leukemia" in another.

Social Networks Derivation. In addition to the explicitly stated friendship network, we extract common interest networks/topic based on tagging patterns. If Joe likes "Pink Floyd" and Jane likes "Madonna", they may both use the tag "music", but will apply it to different items. Therefore, considering the vocabulary is not enough here to determine overlapping interest. Given two users and a topic, a social tie is derived between them if the sets of items they tag with that topic overlap. The added novelty is to use that overlap as a social weight between two users and leverage that information in search and recommendation.

More formally, given a tag t and a user u, we denote by $I(u,t)$ the set of URLs that u tagged with t. Given a topic T and a user u, we denote by $I(u,T)$ the set of resources u tagged with tags $t_i \in T : I(u,T) = \cup_{t_i \in T} I(u,t_i)$. Given two users u_1 and u_2 and a topic T, we say that u_2 is a peer of u_1 for T iff: $|I(u_1,T) \cap I(u_2,T)|/|I(u_1,T)| >$ threshold. Users are peers for a topic if the resources they tag for that topic overlap with a certain confidence. This relationship, like friendship, is directional.

Content Recommendation and Search. In order to enable search and recommendation, we developed instance-optimal algorithms for efficient top-k processing of network-aware content discovery. The "personalized" nature of search introduces the challenge of dynamically computing item scores based on their popularity among members of a social network and the social weights.

Recommender systems are based on looking at correlations between items considered by different users. The added novelty in recommending content in GUESTS comes from the additional tagging information. In GUESTS, recommendations consider both item and interest correlations to focus the recommendation to items derived from the user's social network. For example, a user may tag a site describing the city of New York with "vacation" and share that interest with a colleague. That same user may also tag imdb.com with "movie critic" while the colleague tags it with "actors gossips". Therefore, that colleague's opinion should be considered when recommending other vacation sites to the user and should not matter when recommending a "movie" site.

4. LEVERAGING STRUCTURE

The following projects present research efforts aimed at leveraging existing structure on the web to provide a better

user-experience in various online activities, such as shopping and job search.

4.1 Querying Structured Web Listings

Online shopping has become very popular due to the large inventory of item listings available on the Web. Users can issue a query as a combination of structured and keyword predicates, and the most relevant items are returned in a certain order (e.g., price). We examine two complementary problems when evaluating such queries on listings.

Efficient Evaluation of Top-K Queries over Functions. Very often, online retailers offer price discounts based on promotional rules, e.g., "Stay 3 nights, get a 15% discount on double-bed rooms," or "Buy 2 Motorola Razr cell-phones, get $50 off." Thus, the score for ranking (i.e., the price) is not fixed, but is a function of a parameter in the query, such as the quantity of items being purchased, or the number of nights stayed. We address the problem of efficiently accounting for promotional rules in order to compute dynamic item prices when evaluating queries in online shopping and returning items ranked by price.

We are given a set of items $\mathcal{I}$, a set of parameter values $\mathcal{V}$, and a function $f_i : \mathcal{V} \rightarrow \mathcal{R}$ associated with each item $i \in \mathcal{I}$. For a query $Q = (\texttt{Pred}, v, k)$, where `Pred` is a selection condition on items, $v \in \mathcal{V}$, and k is the desired number of results, we wish to compute a result set R that contains the k lowest-score items that satisfy `Pred`, *where the score of an item i is defined to be $f_i(v)$*[2]. Consider a query Q, `Pred` can be a selection condition on products (e.g., "Make = Canon" and "Color = Blue"), v be the desired quantity of a given product, and k be the number of products that can be shown on the result page.

A naive solution to this problem is to select all the items that satisfy the query predicate, compute the score ($f_i(v)$) for each selected item i, and return the items with the lowest score. Clearly, this approach does not scale to a large number of items and/or unselective predicates. Another simple solution is to precompute and store the score for every (item, value) pair. Queries can then be answered efficiently by simply looking up the top-scored selected items for a given query value. However, the typically large number of items taken in conjunction with many possible parameter values requires space overhead that is particularly bad for large online applications, where all the data and indices are stored in main-memory for efficiency.

To address these limitations, we propose a novel approach where instead of storing the score for every (item, value) pair, we store a compressed representation of this data. We do so by exploiting the fact that the query parameter values are drawn from (or can be mapped to) an underlying ordered domain (e.g., quantity). The key idea is to split the parameter values associated with an item into one or more *intervals*, and then store only the *minimum* score for each (item, interval) pair. The total number of intervals is such that they fit within a specified space budget. We then perform top-k query processing by adapting threshold-based pruning [9, 8] to prune a large number of intervals (and the corresponding items) that cannot possibly make it to the top few results.

For instance, the function "Buy at least 2 items, get 10% off" can naturally be split into two intervals I_1 and I_2. I_1 captures value range $v = 1$ (i.e., $1 \leq v \leq 1$) (assume that the minimum score of f in that range is 150); I_2 captures the value range $v \geq 2$ and the minimum score of f in that range is 0.90×150. Thus, in this example, just by storing two intervals for the item, we obtain a representation that does not lose any information about the function.

The effectiveness of our approach depends on how well the intervals are chosen. One of our main technical contribution is an algorithm that takes as input a given set of items, the corresponding functions, and a space budget, and then uses query workload information to produce a set of intervals that are provably close to optimal for that workload. The algorithm scales linearly with the number of items, and makes few assumptions about the nature of the functions. Specifically, the algorithm only assumes that we can efficiently find the minimum value of f_i (or a relatively tight lower bound of the minimum value of f_i) for a given parameter range, which is true for most rule-based score computations.

Our solution has been tested extensively on large Yahoo! Shopping datasets and shown to be very efficient.

Evaluation of Diverse Query Results. An important but lesser-known concern in online shopping is the ability to return a *diverse* set of results which best reflects the inventory of available listings. For example, a customer in Yahoo! Autos may be interested in finding 5 used 2007 Honda cars. In order to offer the customer the best experience, the system would rather show 5 different Honda models (e.g., Honda Civic, Honda Accord, Honda Odyssey, Honda Ridgeline and Honda S2000) than cars from just one or two models. Similarly, if the user searches for 2007 Honda Civic cars, it would rather show 2007 Honda Civic cars in different colors instead of showing cars of the same color.

The problem of returning a diverse set of query results has been addressed previously. The simplest method is commonly used in search engines: in order to show k results to the user, first retrieve $f \times k$ results (for some $f > 1$) and then pick a diverse subset from these results [4, 16, 17]. Usually, $f \times k$ is much less than the total number of results, so it is more efficient than the previous method. While this method works well in web search, where there are few duplicate or near-duplicate documents, it does not work as well for structured listings because there are many more duplicates. For instance, it is not uncommon to have hundreds of cars of a given model in a regional dealership, or thousands of cameras of a given model in a large online store.

To address the above limitations, we initiate a formal study of the diversity problem in search of methods that are scalable, efficient and guaranteed to produce diverse results. Towards this goal, we first propose a formal definition of diversity, including both unscored and scored variants, that can be used to evaluate the correctness of various methods. We then prove that we cannot use any assignment of static or query-dependent scores to items to implement diversity in an off-the-shelf IR engine (although there is an open conjecture as to whether we can implement diversity using a combination of static and query-dependent scores).

We thus devise evaluation algorithms which implement diversity *inside the database/IR engine*. Our algorithms use an inverted list index that contains item ids encoded using Dewey identifiers. The Dewey encoding captures the notion of *distinct values* from which we need a representative subset in the final query result. We first develop a one-pass algorithm that produces k diverse answers with a single scan over the inverted lists and uses B+-trees to skip over redun-

[2] lower score $\Rightarrow$ higher rank

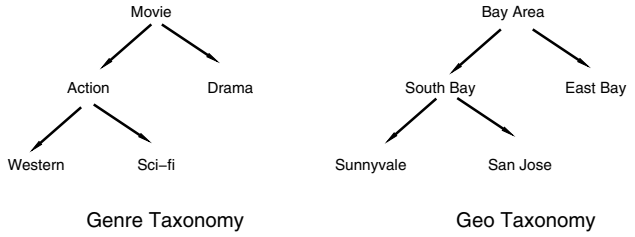

Figure 1: Genre and geo taxonomies.

dant items. We also develop an improved algorithm that is allowed to probe the set of answers within the same distinct value iteratively. The algorithm uses just a small number of probes—at most $2k$. Some of the interesting aspects of our algorithms are that they are provable correct, they can support both non-scored and scored versions of diversity, and they can also support query relaxation (where there may not be enough items satisfying all the query predicates, and hence we relax predicates). Our experiments on large Yahoo! Autos datasets, show that the proposed algorithms are scalable and efficient.

4.2 Search with Taxonomies

The traditional notion of text search deals with a corpus of documents and a query. By building an inverted index for the documents, search can be performed efficiently and documents that "match" the query can be returned to the user. A ranking function is used to quantify the extent of this match and dictate the order of matching documents presented to the user. While this paradigm has been very successful in the field of information retrieval, in many web applications, the above set up is wanting. To illustrate this, consider the following example.

Alice, a resident of Sunnyvale, California, wishes to watch a western action movie in a nearby theater. She queries for such a movie using a search engine capable of supporting local search. The local search engine might take into account the following factors: (1) Alice's query, which is "western action movie"; (2) documents known to the engine, which may be tagged with movie genre such as action or more specifically western action, and a geo location such as Sunnyvale; and (3) Alice's location in Sunnyvale, which is part of South Bay, which is part of the Bay area.

In the above example, item (1) encapsulates the traditional textual query. Items (2) and (3) present a set of desired characteristics of the results to Alice's query; these can be represented as leaves of a genre taxonomy and a geo taxonomy shown in Figure 4.2. Now, imagine the case when there is no movie theater in Sunnyvale that is playing a western action movie, i.e., there are no results to Alice's query that have the genre tag "western action" and the geo tag Sunnyvale. In this case, perhaps, Alice might be desperate enough to drive anywhere in South Bay or even the whole of Bay area to watch a western action movie; this would constitute to generalizing in the geo taxonomy. Alternately, Alice may be content with watching a generic (and not necessarily western) action movie but unwilling to drive outside Sunnyvale; this would constitute to finding results by generalizing in the genre taxonomy. Alice might attach different weights on each of these generalizations.

In our work on search with taxonomies, we precisely capture the above scenario. Formally, we are given a collection of taxonomies $T_1, \ldots, T_m$, where a node in a taxonomy is called a topic and edges can have weights to capture the cost of generalization. Each document in the corpus is associated with exactly one topic in each T_i. The query has two components: a text component (keyword) and a list of m topics, one from each T_i. The answer to a query consists of top k results, ranked in increasing order of the score that is a combination of the text-based relevance score of the document with respect to the query keywords and a total *relaxation cost* for a document with respect to the query topics. The total relaxation cost is the sum over relaxation costs for an individual taxonomy, which is some distance measure between the query topic and the document topic.

In this enhanced retrieval model, we develop new algorithms for indexing and query processing. For indexing, we show how to efficiently encode taxonomy information in an inverted index. For query processing, we decompose the problem into two sub-problems: (i) determining the right level of relaxation for producing the top k results and given the right level (ii) efficiently retrieving documents whose relaxation cost is below the threshold. We provide a complete algorithmic picture of the query processing at fixed relaxation: this problem is solvable for $m = 2$ and NP-hard for $m \geq 3$ but admits efficient approximation algorithms.

4.3 HotJobs

Yahoo! HotJobs is Yahoo!'s online job search tool, bringing together millions of job seekers and job recruiters in an online marketplace with the goal of offering the best possible job seeking/recruiting experience. A collaborative effort between Yahoo! Research and the HotJobs engineering team was initiated a few months back, and was aimed at developing new, state-of-the-art data-analysis tools to further enhance the user experience. In this section, we highlight some of the key issues tackled and progress made during this collaboration, and briefly discuss some of the main outstanding problems for the future. We start with a short description of the current HotJobs environment (circa August 2007).

The HotJobs site offers job seekers the ability to search for relevant job postings using *category* and/or *location*, as well as *keywords* (that are matched against the stored job description). Results are ranked based on relevance, and returned to the user in batches of thirty jobs per result page. The set of categories provided distinguish across jobs at the industry level (e.g., Healthcare, Internet), but not at finer levels (e.g., different job functions within an industry). In addition, a large fraction of job-search queries are quite *vague* (e.g., based solely on category and/or location), resulting in a huge number of results, many of which may not match the user's true intention. In addition, categories can be dominated by postings from a large employer or for a given type of job function (e.g., nurse jobs under Healthcare), which often means that the (most important) first-few search result pages cannot give users a sense of the *diverse* set of results for their query. HotJobs also employs novel Collaborative-Filtering (CF) technology based on the user-job application graph to proactively recommend job postings to registered users who apply for a job. The HotJobs CF-based recommender gives very accurate, focused recommendations (resulting in high user-clickthrough rates); however, it only covers a relatively small fraction of the HotJobs users, that is, registered users who have applied to at least one job. Thus, the user-job application graph is typically sparse, and users

or jobs with no application history cannot benefit from CF recommendations. Based on the above observations, our initial, short-term combined efforts with the HotJobs team have concentrated on two distinct subprojects:

Enhanced CF-based Recommendation Engine. Our key idea here is to broaden the *coverage* of the HotJobs CF tools by drawing user-job affinities based on more than just "apply" edges. More specifically, we propose to enhance the connectivity of the CF graph using the *view history* of users as well as *content-matching* tools. While this is guaranteed to increase the degree of connectivity (and, thus, the coverage) of the HotJobs CF recommender, one potential concern is that it could also decrease the quality of recommendations. Interestingly, initial results with offline testing data show that recommendation quality does not suffer, and, in fact, often *improves* with the added CF connectivity.

Enhanced Job Classification and Diversity Search. We have refined the coarse, industry-level HotJobs categorization of job postings through the use of a hierarchical, content-based document classifier that uses a finer-grained job categorization (e.g., Healthcare(Nursing(NurseStaff, RegisteredNurse, Phlebotomist, ...) ...)). This refined job classification hierarchy obviously enables job seekers to conduct more focused category searches; furthermore, it also allows us to effectively drive *diversity search* algorithms that guarantee the delivery of diversified subsets of results in the first few result pages. Efficiently implementing diversity search using the HotJobs indexing backend posed some interesting research challenges that led us to a novel notion of *approximate* diversity that can be implemented with minimal additional load on a traditional index structure.

One of the key challenges for the future lies in understanding how we can effectively leverage user-behavior to further improve the quality and relevance of the machine-learning tools that support the HotJobs environment. For instance, content-based job classification is a difficult problem, as most of the pertinent information is typically entered as freetext and can often be cluttered by additional text that is not particularly relevent to the job itself (e.g., company profile data). As a result, job descriptions can often be misclassified or placed under several potential categories with low confidence. For such "difficult" jobs, the observed aggregate user-browsing behavior is probably a very useful indicator for deciding the appropriate category, and potentially improving the classification engine itself (through a feedback loop). The design and implementation of such adaptive, self-tuning machine-learning tools that effectively combine content-based features with aggregate user-behavior indicators is a very challenging problem on our research agenda.

5. CONCLUSIONS

The web of tomorrow will likely be significantly different from what it is today. We believe the *systems* aspect of dealing with data at web scale and the *community* aspect of building people-centric applications are keys to the future. This paper describes various ongoing research projects that we are actively pursuing in the Community Systems group within Yahoo! Research based on these convictions.

6. ACKNOWLEDGMENTS

We wish to thank our many collaborators at Yahoo!: Mike Bigby, Bryan Call, Andy Feng, Dan Weaver, and the entire Yahoo! Platform Engineering group; Mani Abrol, K. P. Chitpura, Arun Ramanujapuram, and Srinivasan H Sengamedu of Yahoo! R&D Bangalore; Raj Baskaran, Chris Chen, Adam Hyder, Chris Motes, Geoff Perez, J.P. Samantarai, Abhishek Srivastava, Joe Ting, and Yuan Zhuge at the Yahoo! HotJobs Engineering group; and Lin Guo from Yahoo! Strategic Data Solutions.

We also wish to thank the many academic collaborators: Parag Agrawal from Stanford (PNUTS and Pig); Tyson Condie from UC Berkeley (Pig); Chavdar Botev, Nitin Gupta, and Fan Yang from Cornell (AppForge); Michael Benedikt from Oxford University, Anhai Doan, Pedro DeRose, and Warrent Shen from University of Wisconsin, and Ashwin Machanavajjhala from Cornell (Purple SOX); Julia Stoyanovich from Columbia and Alban Galland from Ecole Polytechnique (GUESTS).

7. REFERENCES

[1] Hadoop. http://lucene.apache.org/hadoop.
[2] Pig Project. http://research.yahoo.com/project/pig.
[3] S. Amer-Yahia, M. Benedikt, and P. Bohannon. Challenges in searching online communities. *IEEE Data Eng. Bull.*, 30(2):23–31, 2007.
[4] J. Carbonell and J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, 1998.
[5] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *SOSP*, 2004.
[6] P. DeRose, W. Shen, F. Chen, A. Doan, and R. Ramakrishnan. Building structured web community portals: A top-down, compositional, and incremental approach. In *VLDB*, 2007.
[7] D. J. Dewitt et al. The gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, 1990.
[8] R. Fagin. Combining fuzzy information from multiple systems. In *PODS*, 1996.
[9] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.
[10] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *SIGMOD*, 2007.
[11] T. Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. *IEEE Data Engineering Bulletin*, 29(1):40–48, 2006.
[12] J. A. Konstan. Introduction to recommender systems. In *SIGIR*, 2007.
[13] R. Ramakrishnan and A. Tomkins. Towards a PeopleWeb. *IEEE Computer*, 40(8):63–72, 2007.
[14] R. Ramakrishnan, A. Tomkins, and R. Kumar. Content, metadata, and behavioral information: Directions for yahoo! research. *IEEE Data Engineering Bulletin*, 29(4):10–18, 2006.
[15] Sandereson and B. Croft. Deriving concept hierarchies from text. In *SIGIR*, 1999.
[16] D. Xin, H. Cheng, X. Yan, and J. Han. Extracting redundancy-aware top-k patterns. In *KDD*, 2006.
[17] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW*, 2005.