Managing Complex and Varied Data with the IndexFabric™

Neal Sample[1,2], Brian Cooper[1,2], Michael Franklin[1,3], Gísli Hjaltason[1], Moshe Shadmon[1] and Levy Cohen[1]

[1]RightOrder Inc.
3850 N. First St.
San Jose, CA 95134 USA

[2]Department of Computer Science
Stanford University
Stanford, CA 94305 USA

[3]Computer Science Division
University of California
Berkeley, CA 94720 USA

{nsample,cooperb}@db.stanford.edu, franklin@cs.berkeley.edu,
{gislih,moshes,levyc}@rightorder.com

Abstract

Emerging networked applications present significant challenges for traditional data management techniques for two reasons. First, they are based on data encoded in XML, LDAP directories, etc. that typically have complex inter-relationships. Second, the dynamic nature of networked applications and the need to integrate data from multiple sources results in data that is semi- or irregularly structured. The IndexFabric has been developed to meet both these challenges. In this demonstration, we show how the IndexFabric efficiently encodes and indexes very large collections of irregular, semistructured, and complex data.

1. Introduction

Java, XML, and the multi-tier architecture for distributed enterprise applications have made the promise of scalability, adaptability, interoperability, and time to market a reality for enterprise computing. The benefits are well known, and this model is quickly becoming a standard. By and large, however, such systems still rely on legacy technologies for data management. These legacy technologies depend on a certain level of predictability, structure, control, and centralization and do not respond well to change and variable structure. As such there is a mismatch between the highly fluid properties of modern applications and the inflexible data repositories used to store and organize the crucial data around which these applications are built.

2. Managing Complex Relationships

Networked applications encompass a wide variety of data items, and these items must be related together in order to answer questions asked by users. For example, in a product catalog, it is not enough to find a single product item; users also want to know specifications for the product, information about the manufacturer, information about vendors, and so on. Relational databases store different entities separately, and then use *joins* to reconstruct relationships among them at query time. The reconstruction of complex objects such as XML documents or LDAP hierarchies can require numerous "self-joins". This reconstruction process is typically quite expensive, and limits the performance of existing systems.

The IndexFabric solves this problem by maintaining data relationships explicitly in an elegant, efficient structure [1,2]. This means that the relationships can be queried and navigated quickly and efficiently. In the IndexFabric, data relationships are explicitly materialized as self-describing entries. This approach supports query formulation and interactive discovery, as the application or user can browse the self-describing structure to determine the relationships that exist among data items.

The IndexFabric represents relationships as *designated strings*. A designator is a special character or string of characters that has semantic meaning. Data items are tagged with an appropriate designator before being inserted into the IndexFabric. For example, a hardware supplier might assign the designator **T** to "item type," **D** to "dimensions" and **P** to "price." Then, a particular item such as a drill can be represented as the keys "**T** Drill [242]", "**D** 11in x 5 in x 7 in [242]", "**P** $64 [242]"; this encodes that item 242 is a drill, with dimensions of 11in x 5 in x 7 in, and which costs $64.

Relationships are then explicitly encoded into strings by concatenating designated items. This flexible approach allows data with varied relationships to be stored uniformly in a single structure. For example, to represent the fact that an item "**T** drill bit [789]" is to be used with another item "**T** drill [988]," the application can create the key "**T** drill [988] **T** drill bit [789]." And insert it into the IndexFabric. To search for drill bits for a particular drill, we can search for keys prefixed by "**T**

drill [988] **T** drill bit." Similarly, a user could discover what types of information are associated with a particular item by using that item as a prefix search key for a range query. For example, a search for keys prefixed by "**T** drill [988]" returns everything related to that drill item, either returning the designators describing related item types, or returning the related items themselves.

3. Balancing Access to Unbalanced Data

Arbitrarily complex relationships can be represented as designated strings, but such strings can become quite long and without special treatment, the costs of management and search could detract from the advantages of the relationship index. Thus, we have developed a system for efficiently storing and querying long relationships in a balanced and predictable fashion.

Our approach uses a novel, block-based implementation of the PATRICIA trie [5]. The PATRICIA trie incurs a fixed cost per key so it scales gracefully while handling long and composite keys. PATRICIA tries, however, are notoriously poor as disk-resident structures. There are two reasons for this. First, n-ary PATRICIA tries are not easily split into block-size pieces leading to poor block utilization, even in bulk-loaded structures. Second, unlike B-tree variants, the shape of a PATRICIA trie depends exclusively on the input dataset. This means that paths through the index may become imbalanced and yield unpredictably many I/Os, even for a single query.

The IndexFabric includes a method for balancing PATRICIA tries to yield predictable, balanced access, while remaining efficient. We refer to the basic unbalanced PATRICIA trie as the "vertical index." Each block in this vertical index can be represented by the longest common prefix of the strings indexed by the block. These prefix strings can be placed into another PATRICIA and parceled out into blocks. This new layer no longer refers to data items, but rather refers to blocks. This process is repeated until there is a PATRICIA that can fit in a single "root block." We show an example of the structure in Figure 1.

Searches begin at the root block and proceed horizontally by reading a single block at each layer, finally entering the vertical index and reaching the data. This balancing structure guarantees predictable search cost and also ameliorates the cost of sub-optimal block utilization. With a block out-degree of about a thousand pointers (given an 8k block size), about a billion keys can be referenced in 3 layers. The upper two layers are less than 10 MB, which easily fit in memory.

Given these features of the IndexFabric, storing keys that represent long and complex relationships is no longer a daunting task. Since each long key adds a constant overhead to the index, relationships may be arbitrarily complex and yet have bounded storage cost. Searching the relationship index is a bounded operation over a small index, due to the aggressive compression of the PATRICIA tries at each level.

This storage approach also facilitates relationship discovery. When generating long keys to represent relationships, closely related elements will have a common prefix. Because the core of the index is a PATRICIA trie, these related elements would be clustered in the index around that common prefix. Thus, when an interesting prefix is located with a single lookup, all encoded related information may be discovered at the same time.

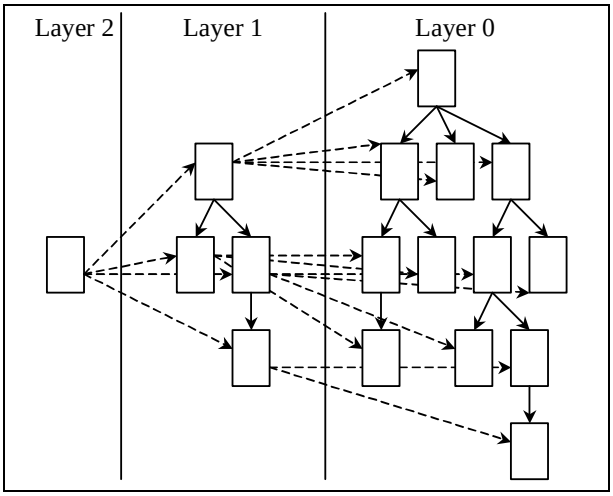

Figure 1. Index Fabric Data Structure

In this demonstration, we will show the techniques used to represent relationships as long keys. We will also show how those long keys behave within a demonstration version of the index. Further information about the technology can be found in [1,2,3,4].

References

[1] B. Cooper, N. Sample, M. Franklin, G. Hjaltason, and M. Shadmon. A fast index for semistructured data. In *Proceedings VLDB*, September 2001, pp 341-350.

[2] Brian Cooper and Moshe Shadmon. The Index Fabric: Technical Overview. Technical Report, 2000. Available at http://www.rightorder.com/technology/overview.pdf.

[3] B. Cooper, N. Sample, and M. Shadmon. A parallel index for semistructured data. *ACM Symposium on Applied Computing 2002*, to appear.

[4] B. Cooper, et al. Extensible Data Management in the Middle-Tier. *Research Issues in Data Engineering (RIDE 2002)*, Feb. 2002, to appear.

[5] D. R. Morrison. PATRICIA - Practical Algorithm to Retrieve Information Coded in Alphanumeric. *Jrnl. of the ACM*, 15(4) pp514-534, Oct 1968.