Application of TRIE data structure and corresponding associative algorithms for process optimization in GRID environment

V. V. Kashansky[a], I. L. Kaftannikov[b]

South Ural State University (National Research University), 76, Lenin prospekt, Chelyabinsk, 454080, Russia

E-mail: [a] vladislav.kash@gmail.com, [b] kil7491@gmail.com

Growing interest around different BOINC powered projects made volunteer GRID model widely used last years, arranging lots of computational resources in different environments. There are many revealed problems of Big Data and horizontally scalable multiuser systems. This paper provides an analysis of TRIE data structure and possibilities of its application in contemporary volunteer GRID networks, including routing (L3 OSI) and specialized key-value storage engine (L7 OSI). The main goal is to show how TRIE mechanisms can influence delivery process of the corresponding GRID environment resources and services at different layers of networking abstraction. The relevance of the optimization topic is ensured by the fact that with increasing data flow intensity, the latency of the various linear algorithm based subsystems as well increases. This leads to the general effects, such as unacceptably high transmission time and processing instability. Logically paper can be divided into three parts with summary. The first part provides definition of TRIE and asymptotic estimates of corresponding algorithms (searching, deletion, insertion). The second part is devoted to the problem of routing time reduction by applying TRIE data structure. In particular, we analyze Cisco IOS switching services based on Bitwise TRIE and 256 way TRIE data structures. The third part contains general BOINC architecture review and recommendations for highly-loaded projects. We suggest some BOINC architectural changes, new ways of TRIE integration to make data processing faster and less dependent on user quantity. We summarize our research highlighting TRIE's benefits and disadvantages, compare it with other data structures like linked lists, hash tables and search trees.

Keywords: Bitwise TRIE, 256-way TRIE, CEF, CFS, volunteer GRID optimization, BOINC

1. Introduction

In computer science TRIE is an ordered multi-way tree type data structure that is used to store byte arrays (strings, structures, dwords, qwords), and especially dictionaries, in extremely effective way. The term trie comes from "re**trie**val". Due to this etymology it is pronounced [tri] ("tree"), although some encourage the use of "try" in order to distinguish it from the more general term. Unlike a search tree, there is no key stored inside a specific node. Instead, node's position in the tree shows what key is associated with it. Each node contains an array of pointers. There is one pointer for each character in the alphabet and all "descendants" of node have a common string prefix associated with that node. Values are normally stored in any node, not only in leaves. The root is associated with the structure's entry point.

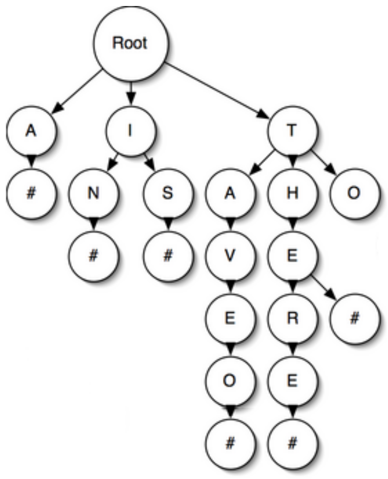

Fig. 1: Example of classic TRIE based storage

Let's consider a classic example, shown on the figure 1. As you can see, root "ancestor" node has an array of pointers to its "descendants" {A, I, T}, one for each allowed alphabet character. All the trie node pointer-fields that don't point to an internal or leaf node are represented using null pointers. To access an information node by key "THERE", we need to move down a series of nodes, extracting characters consequently. Values are represented as "sharp" signs. Values may be simple flags, indicating string existence, or pointers to more complicated structures. By Applying the Big O notation for lookup, insertion and deletion it becomes O(k), which can be written as O(1). Thus these operations are performed in constant time irrespective of the TRIE's items quantity. The constant value only depends on alphabet type, pointer array organization and hardware properties.

2. OSI Layer 3 optimization. Cisco fast switching technology (CFS)

There are many general-purpose structures and algorithms used to build and operate routing tables. The simplest way to search best suitable route for a given IP address is to view (search) records in a table stored in the memory as a linear data structure. For example, a linked list or array. The complexity of the algorithm in this approach will be O(n), where n is a number of entries in the routing table. A similar approach has been implemented in all hardware and "software" routers by default. Of course, this approach is not optimized in terms of search time.

Figure 2 shows a Bitwise TRIE. This structure is considered to store a sequence of N bits as a key. There is no key stored in the intermediate nodes, but an array of pointers is defined. Access to the subsequent elements of specific node is possible to get by index in this array respectively 0 or 1. The access time is O(n), where n is a key length, in the worst case of 32 bits for IPv4.

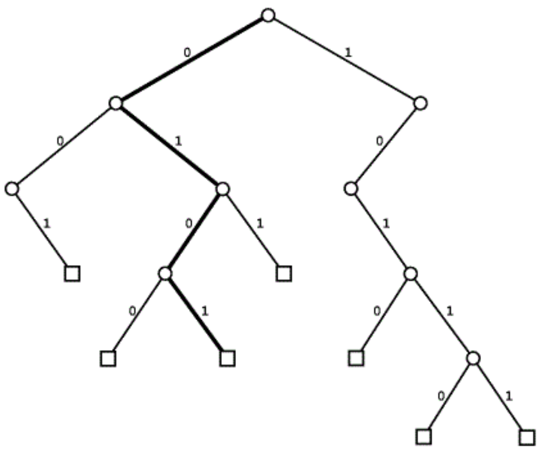

Fig. 2: Storing data in the Bitwise TRIE by 0101 prefix

In CFS technology this structure acts as caching. It requires information from the RIB and ARP tables for its construction, which is then cached in the respective nodes. Traversal is performed on the basis of IPDEST information, extracted from packet. And prefixes are not unique. Therefore, highest priority route, obviously, will be the last extracted (with the largest mask length). The existence of the route is identified by the existence of the ultimate ways of nesting in the tree. There are 32 total permitted (4 * 8 BYTE IPv4) nesting level respectively.

CFS technology highlights:

- This structure allows very fast retrieval of information from the cache. That in turn leads to lower latency;
- The organization of storage requires additional CPU and RAM resources of router;
- When you change the information in the ARP tables, corresponding cache entries should be invalidated and removed.

3. Cisco express forwarding technology (CEF)

Cisco express forwarding (CEF) technology is a further development of CFS technology. It is one of the most progressive and advanced algorithms available today. CEF technology involves using of the **256 way TRIE** data structure known as well as the FIB (Forwarding Information Base). In this approach all opportunities of cache aging are eliminated. 256 way TRIE includes 4 levels of nesting, combining 256 (2^8, size of one IPv4 octet) options at each level. Endpoints of the 4th level contain specific pointers to the data stored in a separate structure. This structure is known as the **Adjacency Table - AJT**. Forwarding Information Base (FIB) or CEF Table is based on the RIB and Adjacency Table. Adjacency Table is based on the ARP table.

Generally CEF was created to optimize the routing processes on large networks and considered to be better than any other switching mode. Usually it is used in backbone routers with a very high load, improving performance of wide area networks. Of course, such improvements in a WAN performance deeply influence volunteer GRID (BOINC based) systems, making it more reliable and fast.

4. OSI Layer 7 optimization. Improvements of BOINC GRID environment

Despite huge popularity among desktop users and the availability of well-known advantages, the BOINC system has a number of unpleasant features. Their negative impact on the system performance increases proportionally to the number of users involved in the calculations, depending on the type of jobs (work units) assigned.

First of all, BOINC combines file server and web server (fig. 3). In fact, uploading and downloading of possibly "heavy" executable and data files (their size usually ranging from a few megabytes to tens of gigabytes) are performed by the Apache web server. This web server is also far from the fastest solution in comparison, for example, with Nginx. This feature can, of course, be ignored when the project has low user base and uses the web server solely for BOINC service purposes, such as file sharing and statistics manipulation via RPC. For highly-loaded projects we strongly recommend to rethink your project architecture in terms of non-blocking socket technologies (select, poll, epoll system calls) to successfully deal with c10k problem. Neglecting it can lead to a serious performance drop down even during early stages of project's lifespan.

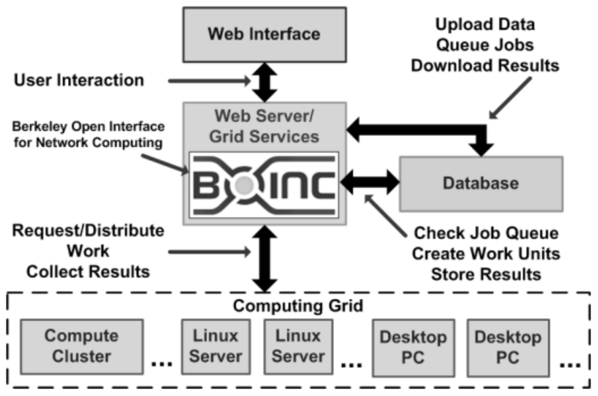

Fig. 3: Architecture of BOINC server-side project

The second fact is that BOINC using Oracle MySQL Server as data storage. It stores the newsfeed, statistics, user data, and other metadata of the entire system. The relational database often becomes a scalability bottleneck and should not be directly accessible by user, especially if you are already suffering from c10k problem.

To achieve maximum performance at different stages of project's lifespan we suggest next approach. Whole project's backend should be divided into two parts: "*lightweight-backend" (next l-b)* and "*heavyweight-backend" (next h-b)*. Really important concept here is the way (*l-b*) accesses the data. Instead of expensive SQL queries to a relational database, (*l-b*) reads the data from a fast non-blocking TRIE key/value storage by default. Meanwhile, (*h-b*) collects all changes from the database and synchronizes them to key/value storage. TRIE could be considered here as engine for NoSQL key/value storage along with other data structures like RB-trees and Hash tables, bringing O(1) complexity. TRIE key/value storage could also be combined with powerful full-text search engine like Elasticsearch for quick full text search, bringing optimal performance for the entire project.

5. Conclusion

Using of TRIE data structure as storage dramatically improved extraction algorithm in terms of access time. Therefore, it's strongly recommended to use TRIE to increase the collision-free efficiency of GRID computational environment. Of course, such a kind of solution is a compromise and should be applied wisely. Despite the fact that optimizations allow fast data manipulation and reduce latency on different OSI layers, it turns into additional resource consumption. More than that, you will have to rethink your system's architecture. Nevertheless, in most cases, when it is required to provide quick access to data records TIRE can be considered as engine for NoSQL servers of GRID environment (OSI L7), as well as for routing storages (OSI L3 CFS,CEF).

Now take a look on the tables 1 and 2 with experimental data. This data was collected during synthetic tests of linked list and bitwise TRIE as storage engines (access operation).

Table 1 – Experimental data output. Linear storage (linked list).
N –entries count, T seconds spent to access

N	2822	8512	13223	19357
T	1.014	2.449	3.742	6.208

Table 2 – Experimental data output. Bitwise TRIE.
N-Key's bits count, T seconds spent to access

N	8	16	24	32
T	0.046	0.086	0.119	0.147

As you can see, using "extraction by key" approach over TRIE storage gives:

- Significant reduction of the access time (see. Table 2). Accessing the elements takes record minimum time. The access time is only proportional to the length of the Key (32 bits for example for CFS);
- Avoid cross-node comparisons of keys and storing them in nodes;
- Avoid tree balancing problem;
- Avoid hash table collisions;
- More efficient spending RAM and CPU resources (compared to other ADT). The CEF technology compared with CFS reduces the load on the CPU and RAM because of more optimal allocation of nodes and key's modified structure.

Meanwhile, using "brute force" over linear storage gives:

- Less RAM resource consumption (data size considered to be the same);
- Search time is proportional to the size of the table;
- Significant consumption of CPU resources, which is proportional to entry count.

References

Robert Sedgewick. Algorithms in C++, Fundamental Algorithms, Data Structures, Sorting, Searching. // "Addison Wesley", 1999.

Donald Knuth. The Art of Computer Programming, Volume 3: Sorting and Searching. Second Edition. // "Addison Wesley", 1998.

Douglas. E. Comer. Internetworking with TCP/IP Principles, protocols and Architecture. // "Pearson Education, Inc.", 2014, 2006, 2000.

How to Choose the Best Router Switching Path for Your Network. [Electronic resource]. URL: http://www.cisco.com/c/en/us/support/docs/ip/express-forwarding-cef/13706-20.html (accessed 20.10.2016).

Cisco IOS Switching Services Configuration Guide. [Electronic resource]. URL: http://www.cisco.com/c/en/us/td/docs/ios/12_2/switch/configuration/guide/fswtch_c/xcfcef.html (accessed 20.10.2016).