

Adaptation of trie hashing for distributed environments.

(Extended abstract)

DR. D.E ZEGOUR

National Computing High School, Algiers

D_zegour@ini.dz

www.multimania.com/zegour

This last decade a new class of data structures, named Scalable Distributed Data Structures (SDDS), was born totally devoted for the multi computers. They are based on client/server architecture. The following properties characterize the SDDS :- Distribution of the file buckets on the servers, - No a main server, - No dialogue between the clients. Each client owns usually an image of the file. The clients can make addressing errors. The client image is updated gradually until obtaining the real image. They are updated using messages called Image Adjustment Messages (IAMs).

Currently, there are two classes of SDDS methods quite distinct : LH-based SDDS and RP-based ones. LH-based SDDS [Lit 93] do not preserve the order of records. They use some information on level to clients to address the file usually distributed on several servers. RP-based SDDS [Lit 94] preserve the order of records. RP*N uses no index and operates simply with multicast. RP*C uses a client index with limited multicast. RP*S uses a servers index with multicast optionally. it was shown that files SDDS can be much faster and larger than traditional files.

Trie hashing (TH) [Lit 81] is one of the fastest access methods for the mono key, ordered and dynamic files. The technique uses a hashing function, which is variable and represented by a digital tree (trie) which grows and retracts according to the insertions and suppressions. Previously, the technique had known much successes. Several works were devoted to it the ten years which followed its creation. Why not to distribute such a powerful file structure? Such is our concern in this paper.

There are several manners of representing the access function generated by TH in memory. The compact representations (CTH) we have suggested before [Zeg 94] make

it possible to double the files addressed by the standard representation (TH) for the same memory capacity. Moreover, for a distributed environment this option is undoubtedly more interesting, in particular for the transfer of the parts of the tree from a site to another, an operation which is frequent in our scheme. The idea of compact representations is to represent the links in an implicit way to the detriment of algorithms of maintenance slightly longer than those of the standard method (TH). The method consumes only 3 bytes per file bucket, what makes it possible to address millions of articles with a small memory capacity.

The question in this paper is to distribute CTH (Compact trie hashing) relatively to the properties of SDDS. We will present the proposed scheme and will show its validity. The results obtained show that the CTH* scheme is promising for the following reasons: - Order-preserving, - Practically without multicast, -Three bytes are sufficient to address a server, - The transfer of some bytes is enough for the updating of trees on the level to the clients.

Below, we describe the distribution of CTH on several sites through an example. Then, we present examples of searching and insertion of keys by various clients. A simulation model is presented followed by a dump screen showing the clients and the servers. Finally, two variants of the method are briefly presented and a conclusion is given.

The concept of scalable distributed compact trie hashing (CTH*)

On the level of each client there is a partial digital tree from which any operation on the file is started. Any client can enter in scene constantly with an empty tree ($| 0$). As symbol ‘|’ designates the largest digit, all the keys are mapped initially in server 0. During the searching phase, commune to all the operations, emanating from a client the tree is updated gradually until obtaining the real tree. On the level of each server there are: - A partial digital tree ; - A bucket containing the records of the file ; - An interval [Min, Max] for this server. The expansion of the file is done through collisions. At each collision there is distribution of the file (splitting of the server) on a logical server. The digital tree of server I is created or extended to each division of server I. It thus keeps the trace of all the splitting on this server. The intervals of the two servers are also

updated. The digital trees are represented in a *preorder* sequential form. The system is initialized firstly only with server 0 with an empty bucket, the largest possible interval [Small, Large] and the tree: | 0. We suppose that Small is the smallest possible key and Large is the largest one. The number of servers is conceptually infinite. The server can be determined in a static or dynamic way. One can have several logical servers for the same physical server.

This new scheme could be similar to the one of the method DRT [KW94].

Illustration Example

The following figure gives the final states of clients and servers after the insertion of the following sequence of the 25 keys (strings) by the corresponding clients (4 clients):

(1 js), (1 hw), (3 c), (2 gwmr), (3 g), (2 km), (4 zur), (1 ewg), (3 lewhv), (2 nrq), (3 mf), (4 pem), (4 rl), (2 bqyg), (3 v), (1 j), (2 qcm), (4 czxav), (2 lhgd), (3 z), (1 lrz), (3 kiyfg), (4 pbtpr), (3 hpqtp), (4 h)

We suppose that the capacity of a bucket (server) is equal to 4.

Clients

1	2	3	4
e0g4k1l2 6	e0g4k1n2r3 5	g0j1k7n2r3 5	g0h1k8n2 3

Servers

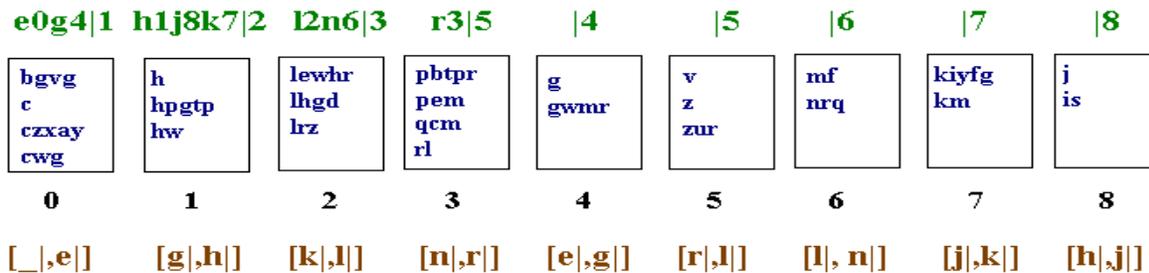


Figure1. Clients and servers

Figure 1. shows 4 clients and 9 servers. Each client has his own access function represented by a digital tree. The file is distributed on 9 servers at a rate of a bucket per server. Each server has an interval indicating the set of the possible keys on this server,

a digital tree keeping the trace of the splitting and a table containing the keys inserted. The client tree represents its image of the file. Thus, for client 1 with digital trie **e 0 g 4 k 1 1 2 | 6** the set of the keys is divided as follows: [Small, e] : 0 ; [e, g] : 4 ; [g, k] : 1 ; [k, l] : 2; [l, Large] : 6. This client thus ‘sees’ only servers 0, 4, 1, 2 and 6. For client 3, with the tree **g 0 j 1 k 7 n 2 r 3 | 5**, there is the following partition: [Small, g] : 0 ; [g, j]: 1 ; [j, k] : 7; [k, n]: 2 ; [n, r] : 3 ; [r, Large] : 5. None of the two partitions is the real partition of the file. If client 1 searches key 'g', it finds server 4. If client 1 searches key 'j', it find firstly server 1. As 'j' is not in the interval of server 1, an addressing error appears. The client then completes its digital tree according to the one of server 1. Its tree becomes **e 0 g 4 h 1 j 8 k 7 1 2 | 6**. The application again of the access function gives us now server 8, i.e. the good. Let us suppose now that client 2 (with the tree **e 0 g 4 k 1 n 2 r 3 | 5**) wants to insert the key 'rym'. The application of its tree to this key gives us server 3. Although the tree of client 2 does not reflect the real image of the file, client 2 finds the good server since key 'rym' is in the interval of this latter. Server 3 cannot contain the new key as its table is full. A collision thus occurs on this server. The collision is solved as follows: one considers the ordered sequence formed of the keys of server 3 and of the new key, that is to say 'pbtp', 'pem', 'qcm', 'rl', 'rym'. Then, we determine the smallest sequence of digits which makes it possible to distinguish the key from the medium ('qcm') and the last key ('rym'). It is thus 'q' this sequence. The tree of server 3 which is **r 3 | 5** thus becomes **q 3 r 9 | 5**. Server 9 is created with an empty tree (| 0) and with the interval [r, Large] . The keys are divided between servers 3 and 9 so that the keys strictly higher than 'q' migrate in the new server. Thus after splitting, the keys 'pbtp', 'pem', and 'qcm' remain in server 3 whereas the keys 'rl' and 'rym' go towards server 9. Let us consider the following case which is a rare event (proven by simulation). Let us try to insert the key 'zz' by client 1. The application of CTH (Transformation Key to a server address) on client 1 (with tree **e 0 g 4 k 1 1 2 | 6**) returns server 6. Key 'zz' does not belong to the interval of this server. As server 6 has an empty tree (| 6) , one is in the presence of a dead end. It is solved by a multicast which returns us server 5 in which will be inserted the key 'zz'. The client tree is updated consequently and becomes **e 0 g 4 k 1 1 2 | 5**. Multicast is also used when a

client meet the Nil node when applying CTH. It has then recourse to the multicasting which delivers to it a server address I. It replaces then Nil by I in its tree.

Simulation Model

Each client is materialized by a table which contains its tree. The servers are compared to the blocks of a file which evolves in a linear way dynamically according to the collisions. A server contains 3 fields: bucket for the keys, an interval and a local tree represented by a table. We considered a very small capacity of the blocks ($b=4$). That makes it possible to have a great number of servers in order to test the validity of the scheme suggested and analyze some parameters. As the load factor is the same as for the other file structures (B-trees, RP, ...) for random insertions, i.e. 70%, then for the same number of servers generated(N_s), the number of inserted articles (N) is given by the formula $N = (B * N_s)*70\%$. Then, the file structure will have the same behavior for the insertion of 500 records with a capacity of block equal to 4 as for the insertion of 500 000 records with a block capacity equal to 4000.

Clients : We considered 4 clients

Servers : Each server is simulated by a file bucket. The capacity of a server : $b = 4$ keys.

Multicasting is simulated by a loop on the existing servers : *For $i := 1$ to N : If Key is in the interval of server S_i : Stop Endfor*

Operations : Construction of a data file with pairs (Client, Key). Clients are randomly generated in set { 1, 2, 3, 4 }. The records are reduced to their keys. Keys are strings randomly generated. A run consists to insert n pairs (client, key) from a data file. We can see the states of clients and servers at the end of a run.

Verification : A program allows to verify the validity of the proposed scheme. A first launch allows to search all the keys inserted by each client. The client trees are then updated. In a second launch we notice no modification on the client trees.

Trace : We can follow the complete trace of the program, What allows us to see the mechanism of the construction of trees and the distribution of the file buckets.

Screen Dump

Figure 2. shows the states of clients and servers after the insertion of 3000 keys with a server bucket capacity equal to 4. 1064 servers are generated with a load factor of 70,49%.

Scalable Distributed Compact Trie Hashing
 Variante 1 : Client Trees, Server trees
 D.E ZEGOUR

Number of keys : 3000
 Server bucket capacity : 4
 Random Insertions

Clients

Client 1

```

a b g 0 335 c a 282 164 d 127 e 72 g f 33 349 k n 280 458 l 183 o c 24 328 q 234 s 110 t 97 u 415 w a 154
320 207 b a t 22 244 b k 149 141 d q 17 481 e m 347 650 g e 144 763 i a 664 297 k 86 l 48 m j 540 332 n
q 43 525 p h 314 264 q 181 s g 613 543 w i 359 235 82 c d f 6 355 f 283 g 123 h 838 j f 115 431 k 412 l
248 m 216 n 592 r g 78 254 t 124 u d 845 520 v 246 x b 817 627 421 d a 69 c 60 d 394 f 373 h a 239 213
l 42 o 89 p 460 q 448 s 354 t 286 u 198 v d 959 686 173 e c h _ 23 620 353 d j 307 270 e 188 f 169 g l
131 278 i b 77 233 k 116 l 848 o 427 p 56 r 317 t 92 v 456 x 341 243 f a 4 c 208 f 143 h n 139 416 i 327 j
226 k c 857 752 l c 737 605 n 68 o 468 p 308 r b 133 105 t 28 u i 515 263 v 261 w 200 x h 273 985 y a
913 478 447 g d 14 f 94 h 30 i t 129 837 j 774 l b 657 o 499 450 m 443 n 351 q i 106 374 r 338 s 249 t g
784 600 v n 163 854 w a 778 476 x j 148 810 305 h b g 20 l 432 250 c 218 e i 117 485 g h 479 310 i e 59
290 k f 186 279 l 161 n 31 o n 240 609 p t 541 746 q m _ 159 847 741 r 670 s 101 t 451 u n 329 762 x
203 y j 166 391 z s 324 798 Nil i a 10 c 95 d 53 e f 529 379 f 291 g 142 h 112 j f 21 503 k i 325 691 n a
288 n 590 437 p 197 r f 90 64 s n 37 677 t m 621 457 u 377 v 236 x o 136 876 y 679 497 j b 9 d b 7 t 507
873 e 449 g 145 i 98 j 466 l b 350 275 m 80 n j 902 683 o k 673 544 q e 358 289 r 287 s 247 u e 177 r
155 711 v 55 w 36 x 333 y d 227 172 114 k a 2 b 710 c 429 d 304 e 147 f t 735 565 g 356 h 202 j h 176
302 k 121 l 360 m 272 n 58 o 571 p m 694 992 q 51 r i 768 312 s 119 t h 587 o 406 109 v a 71 298 w 191
x i 628 977 y 384 502 l b 35 d 316 g 435 i 185 j 156 k 655 m 554 n 260 o 196 p 398 q 83 r u c 680 397
Nil u 182 v 62 w 616 y d 516 674 487 m a 34 e j 16 206 h c 126 660 j h 641 219 m 187 n 54 p 381 q l 296
492 r 210 s 361 t q 281 648 u 594 w 189 x 113 65 n b i 12 531 c 530 d 221 f 532 h m 108 t 473 440 j j
385 602 k b 383 165 m 99 n 85 r 151 t i 132 550 u 439 v t 433 760 w 709 y 313 267 o d 41 e 19 g c 638
483 h 420 i 342 k 301 m 211 q j 96 238 r 179 t 76 v j 403 299 w 268 x q 128 608 y 344 292 p a 8 b 526 c
461 d 346 g 157 h 574 i a 552 508 l 134 o 32 r n 88 583 s 436 u 348 w 306 x 366 y 809 537 q b 27 c c
560 418 d j 404 315 f 258 h 18 j 39 k 199 l 130 m g 564 284 n 26 o 841 q e 724 597 r 362 s 920 u 748 v
494 x 107 y 705 430 r b h l 588 d 393 f 63 i 50 j 146 k j 509 792 l 775 m p 441 409 n 111 p h 47 q 459
916 q 276 r 93 t 75 u 253 v d 715 389 w j 192 644 x 378 z e 175 125 Nil s b h 15 498 c p 472 311 d 225 g
536 i 150 j m 380 506 k 230 m 70 o s 259 523 p i 469 Nil t d 67 528 u o 365 785 v 607 w 424 x 152 496 t
c h 52 556 e f 490 262 g 215 l 49 m 205 o p 477 331 p n 120 517 q 214 s n 84 q 794 623 t 345 v 193 w a
603 405 368 u b 11 d f 194 887 e 174 f t 102 u 557 538 g 352 i l 160 318 j 57 l m 386 321 n 266 p i 445
300 q 222 r 220 u 118 171 v d 5 e 74 f 573 g a 562 255 h 242 i 180 j 45 k e 910 839 l 812 m 716 n c 708
470 o 423 p 326 r k 269 524 s 265 u j 245 728 v h 584 475 x 104 y g 363 964 714 w b 25 c n 190 428 e
241 f 140 h 229 j 46 l a 61 336 m 231 p 178 q 122 r 601 v f 103 488 w 257 x 662 y 504 399 x c c 3 s 293
224 e i 73 661 f 44 g d 549 452 h 390 i 170 j 137 k 758 n 100 o 87 p 66 r b 678 563 t 522 u 251 v 201 w
581 357 y b 40 d 153 e q 81 804 h a 29 542 j 323 l b 757 Nil n d 138 684 p 334 r k 13 p 591 413 v 237 w
162 y a 91 585 387 z c 309 e c 158 395 f b 294 217 g 79 h b 713 465 i 343 j 271 k 671 m 256 r 195 u d
184 y 548 228 v 135 w 277 x 209 z n 501 467 Nil | Nil

```

...

Servers

Server 0

Number of keys : 3

Bucket : aac aafs aagtjhy

Tree : a a g 0 q 1013 878 b g 639 335 c a 282 164 d 127 e 72 l 33 24 b b 22 17 e 6 j 4 q 2 | 1

Interval : >_____ , <=aag|

Server 1

Number of keys: 3

Bucket : raect rajin raxrd

Tree : r a l b h 987 588 d 393 f 63 n 50 47 t 15 u 11 w 5 | 3

Interval : >q| , <=ra|

Server 2

Number of keys: 3

Bucket : kabxsx kag kajw

Arbre : k a j 2 1028 b 710 c 429 d 304 j 147 m 121 p 58 51 l 35 m a 34 16 o 12 | 8

Interval : >j| , <=kaj|

...

Figure 2 : screen dump

Variants of CTH

We can define the two variants :

CTH* with only the Client Trees, i.e. without server trees and without multicast(CTH*CT): On the level of each client there is a partial digital tree from which any operation on the file is started. On the level of each server there are a bucket containing the records of the file and an interval [Min, Max]. There is a central server which contain the real tree.

CTH* Without Server Trees(CTH*WST) : It is the same scheme as that proposed, except that there is no tree on the level of the servers. If the key is not in the interval of the server, one uses multicasting in order to find the good server.

Conclusion

Contrary to the majority of the methods existing, the proposed method provides the order of distributed files and then facilitates the range queries operations and the ordered traversal of files. Moreover, the following properties make of our method a promising opening towards a new class of SDDS : - Order-preserving, - Practically without multicast -Three bytes are sufficient to address a server, - The transfer of some bytes is enough for the update to the trees on the level to the clients, -Access performances should exceed the traditional files and some competitive scalable

distributed data structures.

We proposed the method through only the mechanism of construction. It would be of course more complete to study the other operations in particular the suppression and the range query. Further work should concern the implementation of the communication protocol to test the method in a real network and proceed to deeper analysis of performance.

A basic scheme and two variants have been proposed. According to the application, the one of the scheme can be used : CTH* works with a weak percentage of multicast, almost null. CTH*CT uses the real image of the tree indexing all the servers and thus works without multicast. CTH*WST works with an intense use of multicast but do not uses trees on servers. Further research should concern also many other variants as it was made in LH*. As example, It will be interesting to study the security problem. Now, the method is in its implementation phase under the Unix platform in the context of database query and transaction processing. It constitutes the storage layer of the project 'ACT 21' [Zeg01] which consists to the realization of a parallel data bases management system based on the concept of actors.

Acknowledgements

We would like to express special thanks to M.S Birech, Y. Laalaoui, A. Bekkouche and A. Khaled (students), who contributed by their reflection to the elaboration of some algorithms. A major expression of thanks must go to my dear colleagues W. Litwin and G. Lévy (Paris Dauphine University) for their advice and helpful comments and suggestions.

Some references

- [Lit 81] Litwin, W. Trie hashing. SIGMOD 81. ACM, (May 1981), 19-29.
- [Lit 93] Litwin, W., Neimat, MA., Schneider, D. LH*: A Scalable Distributed Data Structure. (Nov. 1993). Submitted for journal publ.
- [Lit 94] Litwin, W., Neimat, MA., Schneider, D. RP* : A Family of Order Preserving Scalable Distributed Data Structures. Proc. Of 20 th conf. VLDB, chile 1994
- [Zeg 94] D.E Zegour, W. Litwin. Trie hashing with the sequential representations of the trie. International Revue of Advanced Technologies. CDTA, Alger.
- [KW94] : B. Kroll, P. Widmayer. Distributing a search tree among a growing number of processors. ACM Sigmod International Conference On management of Data, May 1994, Sigmod Record 23(2) : 265-276.
- [Zeg01] D.E Zegour. General Presentation of the project 'ACT'. Internal Report. National Computing High School, Algiers, 2001.