An Improved Upper Bound for Scalable Distributed Search Trees *

ADRIANO DI PASQUALE
Università di L'Aquila, Italy

ENRICO NARDELLI
Università di L'Aquila, Italy
Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti", CNR, Italy

GUIDO PROIETTI
Università di L'Aquila, Italy
Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti", CNR, Italy

Abstract

In this paper we analyze the amortized cost of inserts and exact searches in a DRT*, an order preserving scalable distributed data structure able to manage both mono-dimensional and multi-dimensional data. We show that by adding to the DRT* strategy a correction algorithm after split operations, a sequence of m requests of intermixed exact-searches and insertions over a DRT* starting with one empty server and ending with n servers has a cost of $O(m \cdot \alpha(m,n))$ messages, where $\alpha(m,n)$ is the classic inverse of the Ackermann function, thus improving the previous $O(m \log_{(1+m/n)} n)$ bound.

1 Introduction

Scalable Distributed Data Structures (SDDS) [7] are access methods specifically designed to satisfy the high performance requirements of a distributed computing environment made up by a collection of computers connected through a high speed network.

An access method based on the SDDS paradigm has to be *dynamic*: it has to expand to new servers, and only when already used servers are efficiently loaded, and *scalable*: it has to keep the same level of performances while the number of managed objects changes.

The main measure of performance for a given operation in the SDDS paradigm is the number of point-to-point messages exchanged by the sites of the network

*This work has been partially supported by the CNR-Agenzia 2000 Program, under Grants No. CNRC00CAB8 and CNRG003EF8, and by the Research Project REAL-WINE, partially funded by the Italian Ministry of Education, University and Research.

to perform the operation. Neither the length of the path followed in the network by a message nor its size are relevant in the SDDS context. Note that, some variants of SDDS admit the use of multicast to perform range query.

The LH* [7] is the first SDDS that achieves worst-case constant cost for exact searches and insertions, namely 4 messages. It is based on the popular linear hashing technique. However, like other hashing schemes, while it achieves good performances for single-key operations, range searches are not supported efficiently. The same is true for any operation executed by means of a scan involving all the servers in the network.

On the contrary, order preserving structures [1, 2, 3, 6, 8] achieve good performances for range searches and a reasonably low (i.e., logarithmic), but not constant, worst-case cost for single key operations.

In this paper we propose an extension of the DRT* [4, 5], an order preserving SDDS able to manage both mono-dimensional and multi-dimensional data. In [5] we analyzed the amortized behavior of the DRT* and we proved that a sequence of m requests of intermixed exact-searches and insertions over a DRT* starting with one empty server and ending with n servers has a cost of $C(m,n) = O(m\log_{(1+m/n)} n)$ messages. Such a result is obtained by adapting some of the techniques developed for the solution of the Set Union Problem [10] (SUP from now on).

Here we continue to analyze such a relation to find better performances. In fact, we show that by adding to the DRT* strategy a correction algorithm just after each split of a server, we are able to improve the above cost to $O(m \cdot \alpha(m,n))$ messages, where $\alpha(m,n)$ is the classic inverse of the Ackermann function [9]. Due to the well known slow growth of the function $\alpha(m,n)$, we can assume to have $C(m,n) \simeq O(m)$ in realistic scenarios of SDDS made up by thousands or even millions of servers.

The paper is organized as follows: In Section 2 we review basic concepts on the DRT*, in Section 3 we present a sketch of the algorithm and the complexity analysis. Finally, Section 4 concludes the paper.

2 The DRT* data structure

In this section we review the main concepts relative to distributed search trees and in particular to the DRT*, in order to prepare the way for the presentation of our result.

The DRT* (Distributed Random Tree) [5, 6] proposes a distributed search tree for searching both single items and ranges of values in a totally ordered set of keys (allowing insertion of keys). It is basically a search structure, based on key comparisons, managed as a generic tree. The *overall tree* is distributed among the different server sites. Each leaf node is allocated to a different server, together with a partial copy of the overall search structure (called *local tree*). When a leaf node overflows, its bucket is split in two and a new server is brought

in. The overflowing leaf node is transformed in an internal node with two sons (leaf nodes). The leaf node corresponding to new server becomes the root of a new local tree (which is the part of the overall tree allocated to the new server). This node therefore appears twice in the overall tree (once as a leaf in the old, overflowing, node and once as a root in a local tree). Internal nodes are therefore distributed to the different servers according to the way the tree has grown. Client sites query the structure, each using its own view of the overall structure.

A *client view* is a portion of the overall tree, and may be out-of-date since a leaf node may has subsequently been split due to an overflow. A client uses its view to identify to which server the search request has to be sent. If this server evolved and has no more the key, then it forwards the request to the server it identifies using its local trees. This forwarding chain ends at the server having the key.

This last server sends a backward chain of LTC (Local Tree Correction) messages, containing the information about local trees of servers traversed during the forwarding phase, follows the same path followed by the forwarding chain. Information about local trees in an LTC message are used by each server receiving it to update its local tree. The client finally receives, together with the message related to the key, the overall view adjustment.

2.1 Bucket management

The protocol of a server managing a bucket is common to all the proposals on distributed search trees. Each server manages a unique *bucket* of keys. The bucket has a fixed capacity b. We define a server "to be in overflow" or "to go in overflow" when it manages b keys and one more key is assigned to it. When a server s goes in overflow it starts the *split* operation. It requests the address of a new fresh server s_{new} to a special site called *split coordinator*. Whenever s receives the address of s_{new}, it sends to s_{new} half of its keys.

After a split, s manages $\frac{b}{2}$ keys and s_{new} manages $\frac{b}{2}+1$ keys. It is easy to prove the following property:

Lemma 1 *Let σ be a sequence of m intermixed insertions and exact searches. Then we can have at most $\left\lfloor \frac{2m}{b} \right\rfloor$ splits.*

2.2 Local tree

Clients and servers have a local indexing structure, called *local tree*.

The local tree of a client is needed to avoid clients to make address errors. Whenever a client performs a request which results in an *address error*, (i.e., it sends the request to a wrong server), it receives, together with the answer, information to correct its local tree. This prevents a client to commit the same address error twice.

From a logical point of view, the local tree is an incomplete collection of associations $\langle server, interval\ of\ keys \rangle$ managed internally with any data structure: list, tree, etc. For example, an association $\langle s, I(s) \rangle$ identifies a server s and the managed interval of keys $I(s)$. The local tree of a client can be wrong, in the sense that in the reality server s is managing an interval smaller than what the client currently knows, due to a *split* performed by s and yet unknown to the client.

In the DRT*, a client c that wants to perform a request chooses in its local tree the server s that should manage the request and sends it a *request message.* If s is pertinent for the request, then performs it. If s is not pertinent, we have an address error. In this case s looks for the pertinent server s' in its *local tree* and forwards to it the request.

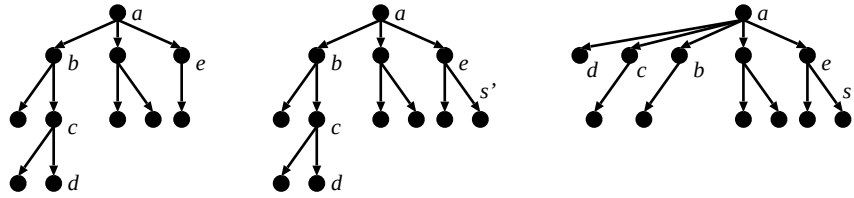
Since s' can be not pertinent as well, it might forward the request to still another server. In general, we can have a series of address error that causes a chain of messages between the servers s_1,s_2,..,s_k. Finally, server s_k is pertinent and can satisfy the request. Moreover, s_k receives the local trees of the servers s_1,s_2,..,s_{k-1} which have been traversed by the request. It first builds a correction tree C aggregating the local trees received and its own one, and then sends LTC messages with C to the client (even if it was an insert operation) and to all servers s_1,s_2,..,s_{k-1}, so to allow them to correct their local trees.

2.3 Split tree

Let T be a DRT*. From the above description of the local trees and how they change due to the distribution of information about the overall structure through LTC messages, it is clear that the number of messages needed to answer a request changes with the increase of the number of requests. To analyze how changes in the content and structure of local trees affect the cost of answering to requests, we associate with each server s of T a rooted tree $ST(s)$, called the *split tree* of s (Figure 1.a shows a split tree). The nodes of $ST(s)$ are the servers pertinent for a request arriving to s. The tree has an arbitrary structure except that the root is s. An arc (s_1, s_2) in $ST(s)$ means that s_1 is in the local tree of s_2. When a split in T occurs, the structure of split trees changes (for example, in Figure 1.b, the split of server e adds the node s' and the arc (s', e) in $ST(a)$).

In the same way, if we consider the correction of local trees, the structure of the split tree of s changes. In fact, due to the correction, after a request to a server d, s adds all the servers in the path between s and d in its local tree. The consequence is that now s can address directly these servers in the future. In order to describe this new situation in the split tree of s, we delete the arcs of the traversed path and add to s the arcs between s and the traversed servers. The result is a compression of the path between s and d (see Figure 1.c).

We use the split trees to takes into account in the amortized analysis the use of LTC messages to reduce the cost of satisfying the request.



(a) (b) (c)

Figure 1: The split tree $ST(a)$ (a). Server e splits, with s' as new server (b). The effect of a compression after a request pertinent for d and arrived to a (c).

3 Correction algorithm after a split

Let us consider a DRT* made up by n servers $s_1, ..., s_n$. Let $ST(s_i)$ be the split tree associated with server s_i, and let h_i be its height. Moreover, let $H = \max\{h_i \mid 1 \leq i \leq n\}$.

Recall that any sequence σ of m requests made up by intermixed inserts and exact searches over a DRT* starting with one empty server and ending with n servers, can be regarded as an instance of a SUP [5]. More precisely, the sequence σ is equivalent to a sequence ρ of m finds, n make-sets and $l \leq n-1$ links. By using the path compression heuristics for the finds, and naive linking for the links, the cost in terms of number of messages of σ is bounded by the cost for executing ρ, that is well known to be $O(m \log_{(1+m/n)} n)$. This is due to the fact there is a correspondence between:

- servers of a DRT* and elements of SUP;
- splits of a DRT* and make-sets and links of SUP;
- split trees of a DRT* and compressed trees of SUP.

Let us now assume for a moment $H \leq c \log n$ for a fixed constant c. In this case we can apply the result in [10], thus obtaining a cost of $O(m \cdot \alpha(m,n))$ for solving the SUP and then for managing a DRT*.

Therefore the idea is to perform some corrections after a split in order to obtain the desired bound. Moreover, the cost of such corrections has to be reasonably low.

3.1 Restricted sequence of splits

In order to describe the correction technique after a split, let us first consider the situation where the last created server is always the next splitting one. In this case, if we do not perform any correction (after either a split or a request) the split tree of any server is always a chain.

We associate with each server s:

- a level $\ell(s) \geq 1$, where $\ell(s) = 1$ when s is just a new fresh server.
- a pointer $p(s)$ to the *parent server* of s. After the split of s with t as new server, we set $p(t) = s$. Notice that $p(s_0) = NULL$, where s_0 is the initial server of a DRT*.

We recall that a *correction* between two servers s and t consists of the following steps: t sends a copy of its local tree to s; s updates its local tree with the one received by t (see [5] for more details).

We define a *k-correction at level* λ, starting from server s_j, as a sequence of correction between s_i and s_{i-1}, for $i = j, ..., j-k+1$. Moreover, after the correction between s_{i+1} and s_i, if s_i has level λ, then it updates its level to $\lambda+1$, for any $j-k \leq i \leq j-1$. After that, s_{j-k} sends a message to servers $s_{j-k+1}, ..., s_{j-1}, s_j$ in order to update their parent pointer to s_{j-k}. A k-correction at level λ costs $2k$ messages.

We define the *k-correction technique* as follows. Suppose we do not operate any correction through the first $k-1$ splits and the k-th split has just occurred. Therefore, we have a chain of length k of servers $s_0, ..., s_k$ (see Figure 2.a). After the k-th split we apply the first k-correction starting from server s_k. That is, server s_k sends a copy of its local tree to server s_{k-1}. Server s_{k-1} updates its local tree with the one received by s_k, updates its level to 2, and then sends a copy of its local tree to s_{k-2}. The operation continues up to s_0 (see Figure 2.b), and then s_0 sends the messages to update the parent pointer of servers $s_1, ..., s_k$. Now, the level of servers $s_0, ..., s_k$ is 2 and the parent pointer of servers $s_1, ..., s_k$ is set to s_0 (see Figure 2.c). After that, we wait for the next k splits, and we then perform another k-correction at level 1 starting from s_{2k} and involving the servers $s_k, ..., s_{2k}$. After k corrections at level 1, the server s_{k^2} starts the first k-correction at level 2 involving the servers $s_0, s_k, s_{2k}, ..., s_{k^2}$ (see Figure 3.a). After that, the level of servers $s_0, s_k, s_{2k}, ..., s_{k^2}$ is 3. Figure 3.b shows $ST(s_0)$ and $ST(s_1)$ after the k-correction at level 2. Figure 6 shows $ST(s_0)$ just before s_{k^λ} starts a k-correction at level λ.

In the following we denote with T a DRT* starting with one server s_0 and where the next splitting server is always the last created one. We apply to T the k-correction technique for a given $k > 1$. Let $h_j = h(ST(s_j))$ denote the height of $ST(s_j)$ and let H be the height of T, that is $H = \max_{s_j \in T} \{h_j\}$. In the following we show that with a suitable choice of k, H has a logarithmic bound and then the result in [10] can be applied.

In order to prove the result we give a set of definitions. We say T is at level λ if the server with the maximum level has level λ. We want to compute the longest height of T when T is at level λ.

Moreover, let A_v represent a tree with a root node connected to k^v nodes, for a given k (see Figure 4). Let $F_\lambda(s_0)$ represent the shape of $ST(s_0)$ just after the

k-correction at level $\lambda - 1$. $F_\lambda(s_0)$ represents $ST(s_0)$ when T is at level λ for the *first* time.

Lemma 2 *$F_\lambda(s_0)$ has the shape of $A_{\lambda-1}$, for a given k and for $\lambda > 1$.*

Proof.

It is easy to see that the claim is true for the case $\lambda = 2$ (see Figure 4).

Suppose by induction $F_\lambda(s_0)$ has the shape of $A_{\lambda-1}$, for a given $\lambda > 1$. Considering the *k*-correction technique and that the next splitting server is always the new created one, it is easy to show that the *k*-correction at level $\lambda + 1$ starts with the configuration of Figure 5.a. After that we have the configuration of Figure 5.b. By definition this is $F_{\lambda+1}(s_0)$ and it corresponds to A_λ (Figure 5.c).

□

Let $L_\lambda(s_0)$ represent the shape of $ST(s_0)$ just before the *k*-corrections leading T at level $\lambda + 1$. It is easy to show with a proof by induction similar to the one of Lemma 2 that Figure 6 represents $L_\lambda(s_0)$. In fact, after the last split, a *k*-correction at level 1 would start, followed by a *k*-correction at level 2 and finally followed by a *k*-correction at level λ. $L_\lambda(s_0)$ represents $ST(s_0)$ when T is at level λ for the *last* time.

Lemma 3 *Consider T at level $\lambda > 1$ when $ST(s_0)$ corresponds to $F_\lambda(s_0)$. Let H be the height of T. Then $H = h(ST(s_1)) = \lambda - 1$.*

Proof.

It is easy to see that the claim is true for $\lambda = 2$ (see Figure 4).

Suppose by induction $H = \lambda - 1$, for a given $\lambda > 1$. Consider the *k*-correction at level λ starting from the configuration shown in Figure 5.a. Due to the correction, the distance between server $s_{k^{\lambda-1}}$ and s_{k^λ} in $ST(s_{k^{\lambda-1}})$ is just one. Hence, it is easy to show that the height h_1 is the distance between server s_1 and $s_{k^{\lambda-1}}$ in $ST(s_1)$, plus the distance between server $s_{k^{\lambda-1}}$ and s_{k^λ} in $ST(s_{k^{\lambda-1}})$. But, by induction the distance between server s_1 and $s_{k^{\lambda-1}}$ in $ST(s_1)$ is equal to $\lambda - 1$, hence $h_1 = \lambda$. It is easy to show that any other split tree has a smaller height. Therefore, after the *k*-correction at level λ, $H = h_1 = \lambda$, and T is at level $\lambda + 1$ when $ST(s_0)$ corresponds to $F_{\lambda+1}(s_0)$. □

Lemma 4 *Consider T at level $\lambda > 1$. Let h_0 be the height of $ST(s_0)$. Then $h_0 \leq (\lambda - 1)(k - 1) + k$, for a given choice of k.*

Proof.

Consider T at level $\lambda > 1$ and suppose we apply the *k*-correction technique to T, for a given *k*. We show that h_0 is maximum when $ST(s_0)$ has the shape of $L_\lambda(s_0)$.

This is true for $\lambda = 2$ (see an example for $k = 4$ in the third box of Figure 3.a).

Suppose by induction the above statement is true for a given $\lambda > 1$. Consider now T is at level λ. Suppose it is the first time T is at level λ and then $ST(s_0)$ has the shape of $F_{\lambda+1}(s_0)$. Further split from $s_{k^{\lambda-1}}$ build up $ST(s_{k^{\lambda-1}})$ as a subtree of $ST(s_0)$. Due to the k-correction technique, $ST(s_{k^{\lambda-1}})$ evolves in the same way as $ST(s_0)$. Hence, by the inductive hypothesis, the height $h_{k^{\lambda-1}}$ of $ST(s_{k^{\lambda-1}})$ is $h_{k^{\lambda-1}} \leq (\lambda-1)(k-1)+k$ and we have the equality whenever $ST(s_{k^{\lambda-1}})$ has the shape of $L_\lambda(s_{k^\lambda})$. In such a configuration, $h_0 = 1 + h_{k^{\lambda-1}} = 1 + (\lambda-1)(k-1)+k$ and it is the maximum height h_0 has reached. Due to the fact that $ST(s_{k^{\lambda-1}})$ has the shape of $L_\lambda(s_{k^{\lambda-1}})$, we have a set of k-corrections leading $ST(s_{k^{\lambda-1}})$ to the shape of $A_{\lambda+1}(s_{k^{\lambda-1}})$. In such new configuration, $h_0 = 1+1$. Now we can apply the previous argument to the server $s_{2k^{\lambda-1}}$. Upon $ST(s_{2k^{\lambda-1}})$ has the shape of $L_\lambda(s_{2k^{\lambda-1}})$, $h_0 = 1+1+h_{2k^{\lambda-1}} = 2+(\lambda-1)(k-1)+k$ and it is the maximum height h_0 has reached. It is straightforward that the maximum height h_0 can reach when T is at level $\lambda+1$ is when $ST(s_0)$ has the shape of $L_\lambda(s_0)$ (see Figure 6). In this case

$$h_0 = (k-1) + h_{(k-1)k^{\lambda-1}} = (k-1) + (\lambda-1)(k-1) + k = \lambda(k-1) + k.$$

□

Lemma 5 *Consider T at level $\lambda > 1$. Let H be the height of T. Then $H \leq \lambda k - 1$, for a given choice of k.*

Proof.

Consider T at level $\lambda > 1$ and suppose we apply the k-correction technique to T, for a given k.

Following a proof by induction like in lemma 4 it is possible to show that H is maximum when $ST(s_0)$ has the shape of $L_\lambda(s_0)$ and that $H = h_1$.

Consider such a configuration. From Lemma 4, $h_0 = (\lambda-1)(k-1)+k$. From Lemma 3, the distance between s_1 and $s_{k^{\lambda-1}}$ in $ST(s_1)$ is $\lambda - 1$. Hence

$$h_1 = (\lambda-1) + (h_0 - 1) = (\lambda-1) + (\lambda-1)(k-1) + k - 1 = \lambda k - 1.$$

It is easy to show that any other split tree in T has a smaller height.

□

Let us now compute the relation between the level of T and the number of splits in T.

Lemma 6 *Let λ be the final level of T. Let n be the number of splits in T. Then, for a given choice of k, $\lambda \leq \lfloor \log_k n \rfloor + 1$*

Proof.

We perform $\left\lfloor \frac{n}{k} \right\rfloor$ corrections at level 1, $\left\lfloor \frac{n}{k^2} \right\rfloor$ corrections at level 2, and finally we have $1 \leq \left\lfloor \frac{n}{k^{\lambda-1}} \right\rfloor \leq k-1$ corrections at level $\lambda - 1$ leading T at level λ. In this case, $\lambda \leq \lfloor \log_k n \rfloor + 1$. □

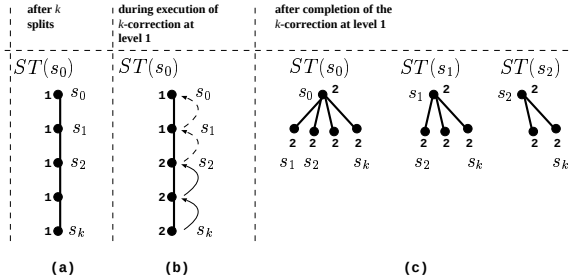


Figure 2: A sequence of k splits (a), the k-correction at level 1 (b) and the final configuration of $ST(s_0)$, $ST(s_1)$ and $ST(s_2)$ (c).

The following result is a direct consequence of previous lemmas.

Theorem 1 *Let H be the height of T. Then $H \leq k \log_k n$, for a given choice of k.*

Finally, if we choose $k = 2$, we have $H \leq 2 \log_2 n$.

Now, we compute the cost in terms of messages of applying the the k-correction technique to T.

Lemma 7 *Let λ be the final level of T. Let n be the number of splits in T. The number of messages for the correction technique is $M < 4n$.*

Proof.

We perform $\left\lfloor \frac{n}{k} \right\rfloor$ corrections at level 1, $\left\lfloor \frac{n}{k^2} \right\rfloor$ corrections at level 2, and finally we have $1 \leq \left\lfloor \frac{n}{k^{\lambda-1}} \right\rfloor \leq k-1$ corrections at level $\lambda - 1$. Hence

$$M = \left\lfloor \frac{n}{k} \right\rfloor 2k + \left\lfloor \frac{n}{k^2} \right\rfloor 2k + \ldots + \left\lfloor \frac{n}{k^{\lambda-1}} \right\rfloor 2k \leq 2n \left(1 + \frac{1}{k} + \ldots + \frac{1}{k^{\lambda-2}} \right) \leq 4n.$$

□

3.2 Generic sequence of splits

In the following we denote with T a DRT* starting with one server s_0. We apply to T the k-correction technique for $k = 2$. Figure 7 shows an example of the configuration of T for $k = 2$. Let us assume a generic sequence of splits. In this case, we have a tighter bound on H, because we perform more corrections with respect to the chain. Hence, the previous bound on H of Theorem 1 holds.

It remains to calculate the cost in terms of number of messages of the correction technique.

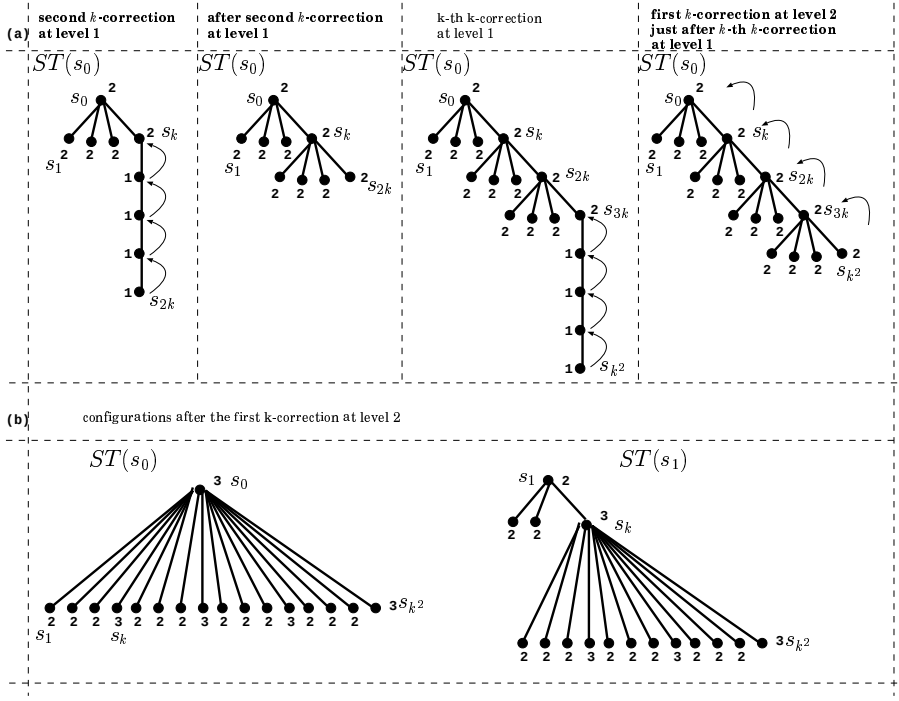


Figure 3: Evolution before the first k-correction at level 2 (a). $ST(s_0)$ (left) and $ST(s_1)$ (right) after the k-correction at level 2 started at s_{k^2} (b).

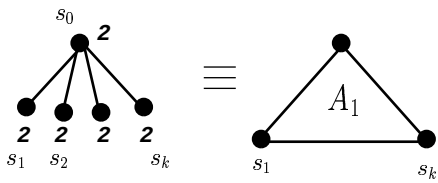


Figure 4: The tree A_1 for a given k.

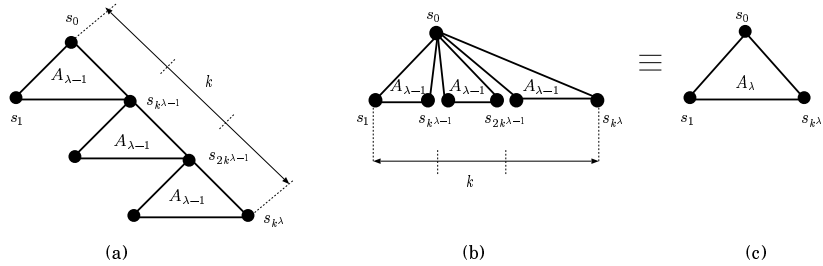


Figure 5: $ST(s_0)$ before (a) and after (b) the k-correction at level λ leading T at level $\lambda+1$.

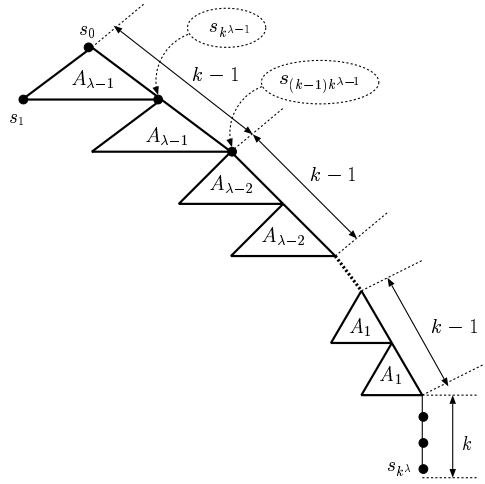


Figure 6: $ST(s_0)$ of a DRT* at level λ, just before the k-correction of s_{k^λ}.

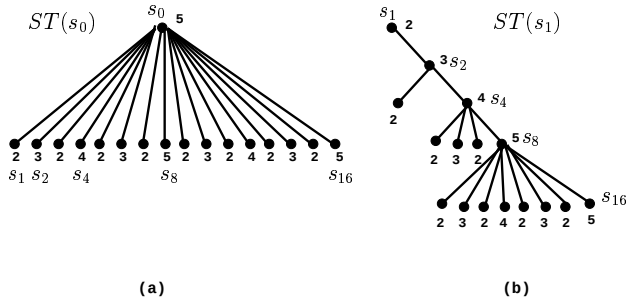


Figure 7: $ST(s_0)$ (a) and $ST(s_1)$ (b) after a k-correction at level 4, with $k=2$

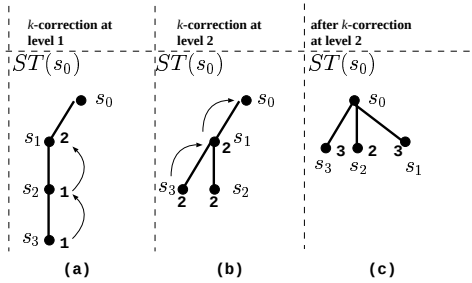


Figure 8: The number of *k*-corrections with respect to the number of splits starting from a server at level 2.

Lemma 8 *Let n be the number of splits in T. The number of messages for the correction technique is $M = 4(n-1)$.*

Proof.

Let us consider Figure 8.a. Suppose a chain of splits starts from s_1. After 2 splits we have a 2-correction at level 1 (Figure 8.a), followed by another 2-correction at level 2 (Figure 8.b), because s_1 is at level 2 and it is not s_0. The situation is now shown in Figure 8.c.

Suppose a chain of splits starts from s_2. After 2 splits, we have a 2-correction at level 1. After the next 2 splits, we have a 2-correction at level 1, followed by another one at level 2 and another one at level 3, because s_2 is at level 3 and it is not s_0.

The same argument applies to s_4, s_8, s_{16}. Hence if we start from a server at level 1, we get a 2-correction after just 1 split and we get no 2-correction after 1 split from a server at level greater than 1. If we start from a server at level 2, after 2 splits we get at most two 2-corrections. If we start from a server at level 3, after 4 splits we get at most four 2-corrections. In general if we start from a server at level i, after 2^{i-1} splits the number of 2-corrections is at most 2^{i-1}.

Since T starts with s_0 at level 1, to get a 2-correction we need at least 2 splits. Then, after 2 splits, we have at most a 2-correction at level 1. After that, all the servers are at level 2. It is clear that if we always perform a chain of 2 splits starting from a server at level 2, we get the maximum number of 2-corrections per splits. The associated number of messages is then $M = 4(n-1)$, where n is the number of splits, because each 2-correction needs 4 messages.

□

Finally, from Theorem 1, Lemma 1, Lemma 8 and from the result in [10] we have our main result:

Theorem 2 *Let T be a DRT* starting with one empty server and ending with n servers. If we apply to T the k-correction technique for $k = 2$, then the number of*

messages of a sequence of m requests made up by intermixed inserts and exact searches over T is

$$C(m,n) = O(m \cdot \alpha(m,n)) + 8\frac{m}{b} = O(m \cdot \alpha(m,n)).$$

4 Conclusions

We considered the DRT*, a classical SDDS able to manage both mono-dimensional and multi-dimensional data. We studied the variant of the DRT* to which the *k-correction technique* is applied. Basically, this technique is based on a policy of standard corrections after splits of servers in the DRT*.

We showed that, combining the *k*-correction technique and the standard correction technique after an address error, a sequence of *m* requests of intermixed exact-searches and insertions over a DRT* starting with one empty server and ending with *n* servers has a cost of $O(m \cdot \alpha(m,n))$ messages, where $\alpha(m,n)$ is the classic inverse of the Ackermann function. Due to the well known slow growth of the function $\alpha(m,n)$, we can assume to have amortized constant costs for inserts and exact searches in realistic scenarios of SDDS made up by thousands or even millions of servers. The result appears very promising in the light of the possibility of the DRT* of managing multi-dimensional data and of good performance for multi-keys requests, typical of order preserving SDDS.

References

[1] P. Bozanis, Y. Manolopoulos: DSL: Accomodating Skip Lists in the SDDS Model, *Workshop on Distributed Data and Structures (WDAS 2000)*, L'Aquila, June 2000.

[2] Y. Breitbart, R. Vingralek: Addressing and Balancing Issues in Distributed B^+-Trees, *1st Workshop on Distributed Data and Structures (WDAS'98)*, 1998.

[3] A.Di Pasquale, E. Nardelli: Fully Dynamic Balanced and Distributed Search Trees with Logarithmic Costs, *Workshop on Distributed Data and Structures (WDAS'99)*, Princeton, NJ, May 1999.

[4] A.Di Pasquale, E. Nardelli: An Amortized Lower Bound for Distributed Searching of *k*-dimensional data, *Workshop on Distributed Data and Structures (WDAS 2000)*, L'Aquila, Carleton Scientific, June 2000.

[5] A.Di Pasquale, E. Nardelli: Distributed searching of *k*-dimensional data with almost constant costs, *ADBIS 2000*, Prague, Lecture Notes in Computer Science, Vol. 1884, pp. 239-250, Springer-Verlag, September 2000.

[6] B. Kröll, P. Widmayer: Distributing a search tree among a growing number of processor, in *ACM SIGMOD Int. Conf. on Management of Data*, Minneapolis, MN, 1994.

[7] W. Litwin, M.A. Neimat, D.A. Schneider: LH* - Linear hashing for distributed files, *ACM SIGMOD Int. Conf. on Management of Data*, Washington, D. C., 1993.

[8] W. Litwin, M.A. Neimat, D.A. Schneider: RP* - A family of order-preserving scalable distributed data structure, in *20th Conf. on Very Large Data Bases*, Santiago, Chile, 1994.

[9] R.E. Tarjan, Efficiency of a good but nonlinear set union algorithm, *J. Assoc. Comput. Mach.*, 22,2 (1975), pp. 215-225.

[10] J. Van Leeuwen, R.E. Tarjan, Worst-case analysis of set union algorithms, *J. Assoc. Comput. Mach.*, 31,2 (1984), pp. 245-281.

Adriano Di Pasquale is with the Dipartimento di Informatica, Università di L'Aquila, Via Vetoio, Coppito, I-67010 L'Aquila, Italy. E-mail: dipasqua@di.univaq.it

Enrico Nardelli is with the Dipartimento di Informatica, Università di L'Aquila, Via Vetoio, Coppito, I-67010 L'Aquila, Italy and with the Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti", Consiglio Nazionale delle Ricerche, Viale Manzoni 30, I-00185 Roma, Italy. E-mail: nardelli@univaq.it

Guido Proietti is with the Dipartimento di Informatica, Università di L'Aquila, Via Vetoio, Coppito, I-67010 L'Aquila, Italy and with the Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti", Consiglio Nazionale delle Ricerche, Viale Manzoni 30, I-00185 Roma, Italy. E-mail: proietti@univaq.it