An RP* Extension with Amortized Almost Constant Cost

Adriano Di Pasquale .... University of L'Aquila, I.Z.S.A.M.

Guido Proietti .... University of L'Aquila, I.A.S.I. - CNR

Enrico Nardelli .... University "Tor Vergata" Rome, I.A.S.I. - CNR

Outline of the talk

- A brief SDDS introduction.
- Order Preserving SDDSs.
- The “Split Tree”: a tool to analyze Order Preserving SDDSs.
- RP*plus:
    - The search process.
    - The analysis of RP*plus using “Split Trees”.
    - High concurrency environment.
- Conclusions.

What is an SDDS

- A paradigm used to define new type of data structure to manage distributed files.
  - Specifically for multicomputers
- Designed for high-performance files.
  - Scalability to very large sizes
    - Performance independent from the size of the file.
    - larger than any single-site file
  - processing in (distributed) RAM
  - access time better than for any disk file
  - Parallel & distributed queries
  - Distributed autonomous clients

Applications

- Object and Relational Databases
- WEB servers
- Video servers
- Real-time systems
- Scientific data processing

SDDS

Assumptions

- Set of servers managing the domain of values, with capacity *b*.
- Set of clients using items with an arbitrary behavior.
- Point-to-point protocol.
- Network is reliable: both nodes and links never fails.

Properties

• Dynamic:	Number of servers changes while number of objects changes.
• Scalable:	No centralized control. Performance independent from the size of the file.
• Local:	Perform operations locally.

Performance Measures

- **Access cost** (key & range searches, Inserts, etc.)
  - number of messages
    - Network independent
    - Size and route of a message are not accounted (network topology is a complete graph)
  - access time
- Storage cost
  - load factor
    - same definitions as for the traditional Data Structures

SDDS(contd)

Requests: Clients perform requests. They have a local index to find the correct server for a request.

Split: Whenever a server *s* is in overflow half of its objects is transferred to a new server *s'*. This is a local operation.

Address error: A client can send a request to a server *s* for an object transferred to another server, due to a split.

Index correction: Using information collected in the management of the address error, the client's view (server's view) of the file distribution is updated.

Mono-dimensional SDDS

Operations: exact-search, range-search, insertion, deletion.

LH* (Litwin, Neimat, Schneider: SIGMOD 1993)

Distributed version of Linear Hashing.

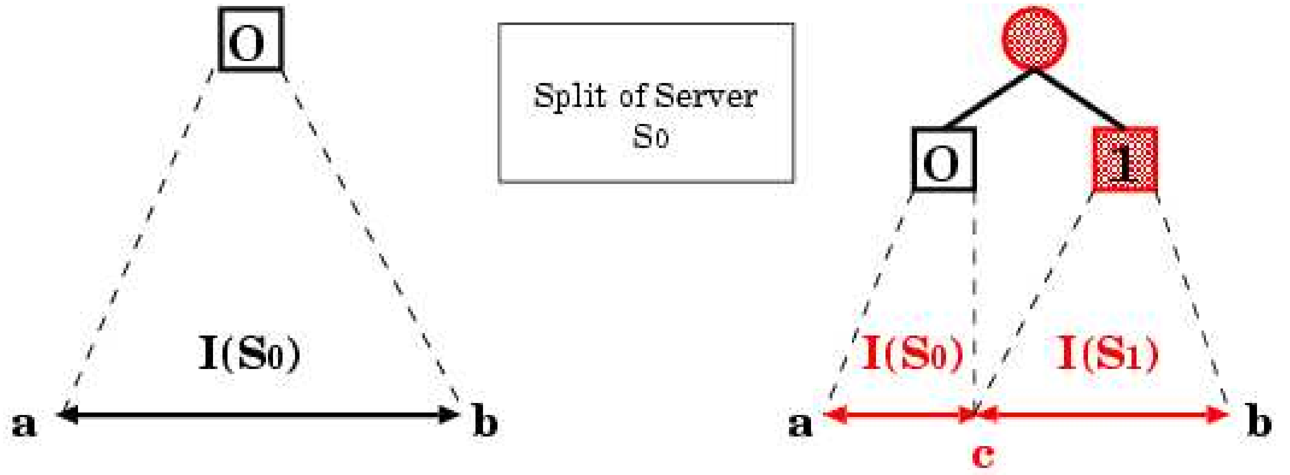
- 4 messages for an exact-search and an insertion (W.C.).
- *O*(*n*) messages for a range-search (W.C.).

Order Preserving SDDSs

- Designed to obtain good range search performances.

Order Preserving SDDS

- **Split:** interval partitioning and creation of a virtual internal node.
- **Local Trees:** servers also manages internal nodes.

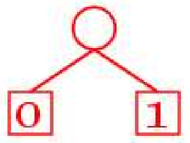
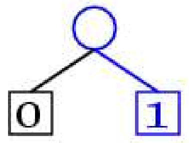


Fixed Index:	• **BDST** (Di Pasquale, Nardelli: WDAS 1998) • **RP*** (Litwin et al. : VLDB 1994)
Incremental Index:	• **DRT** (Kroll, Widmayer:SIGMOD 1994) • **DRT*** (Di Pasquale, Nardelli: ADBIS 00, WDAS 00-02) • **Distributed B$^+$-tree**. (Breitbart, Vingralek:WDAS 98) • **ADST** (Di Pasquale, Nardelli: DEXA 01, SOFSEM 01)

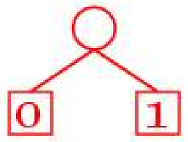
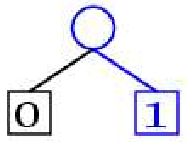
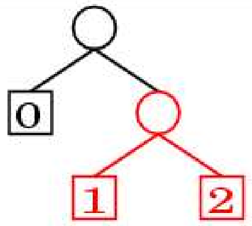
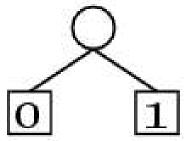
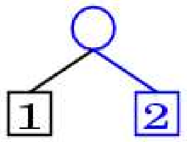
Order Preserving SDDS

	Virtual Tree	T(S_0)	T(S_1)	T(S_2)	T(S_3)
	$\boxed{0}$	$\boxed{0}$			

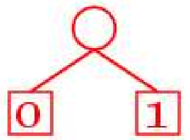
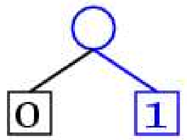
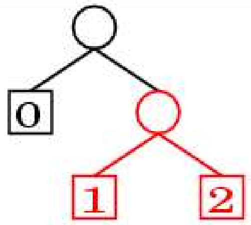
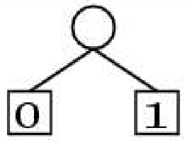
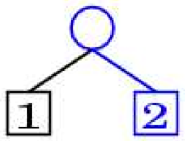
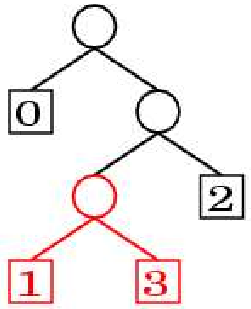
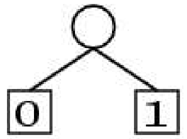
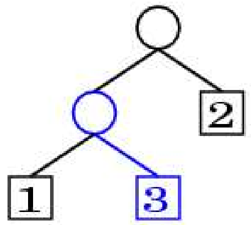
Order Preserving SDDS

	Virtual Tree	$T(S_0)$	$T(S_1)$	$T(S_2)$	$T(S_3)$
	$\boxed{0}$	$\boxed{0}$			
After split of S_0			$\boxed{1}$		

Order Preserving SDDS

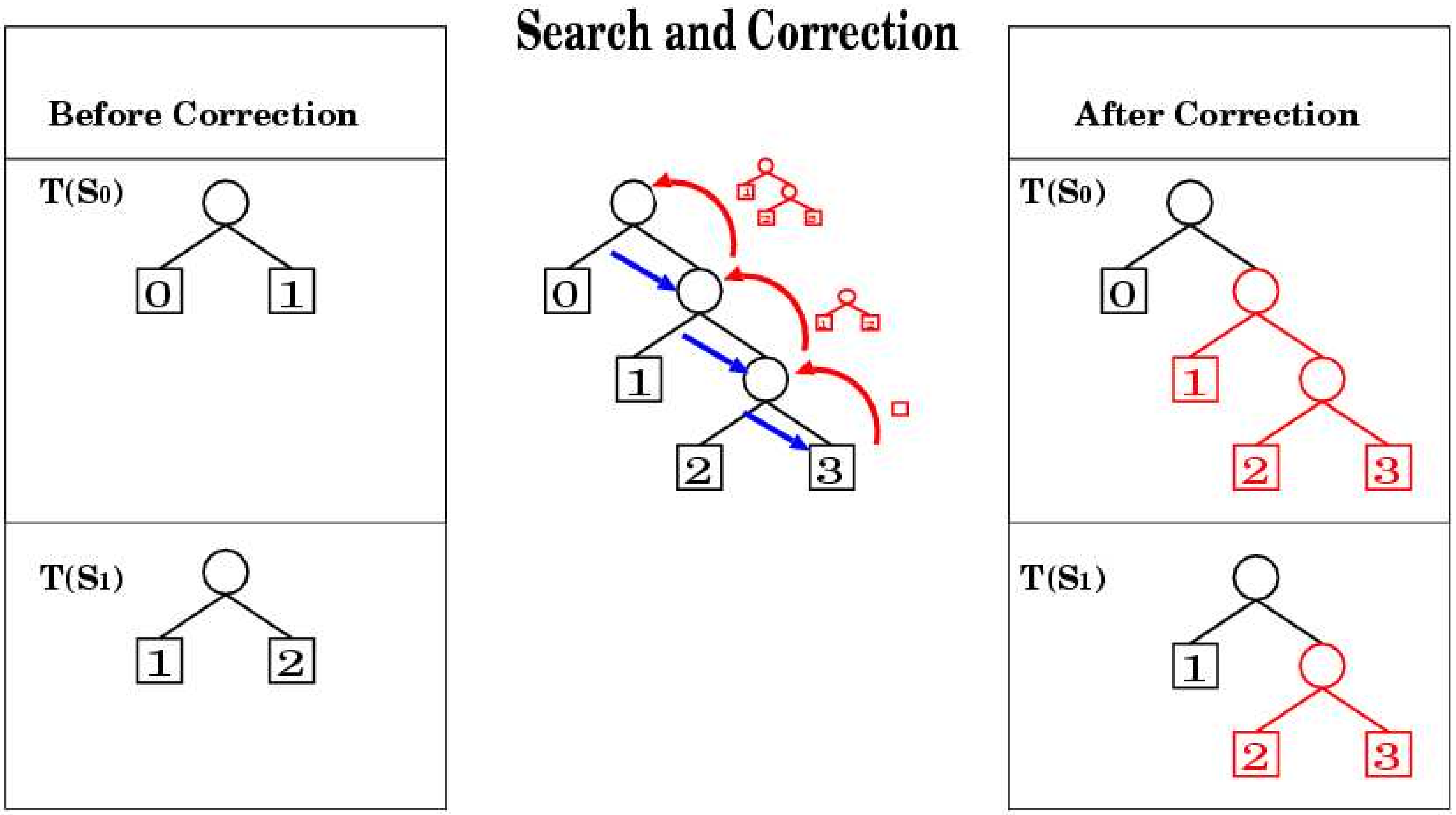
	Virtual Tree	T(S0)	T(S1)	T(S2)	T(S3)
	$\boxed{0}$	$\boxed{0}$			
After split of S0			$\boxed{1}$		
After split of S1				$\boxed{2}$	

Order Preserving SDDS

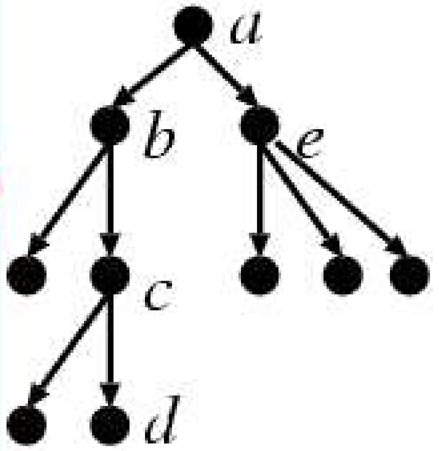
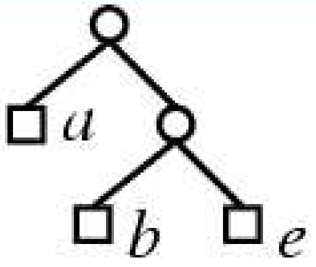
	Virtual Tree	T(S_0)	T(S_1)	T(S_2)	T(S_3)
	$\boxed{0}$	$\boxed{0}$			
After split of S_0			$\boxed{1}$		
After split of S_1				$\boxed{2}$	
After split of S_1				$\boxed{2}$	$\boxed{3}$

DRT & DRT*

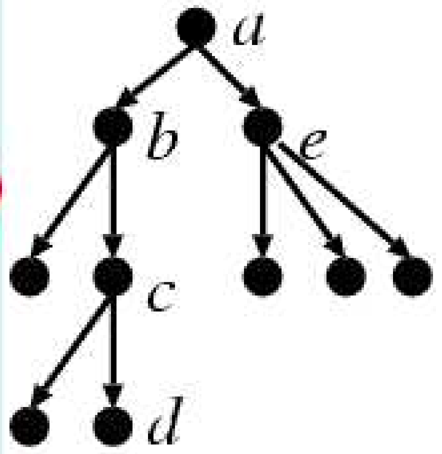
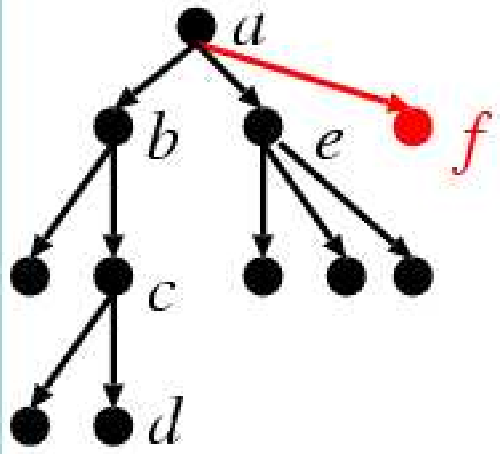
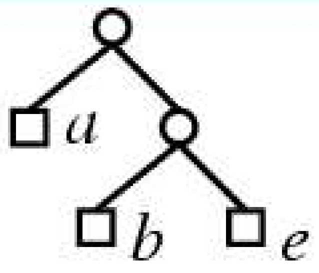
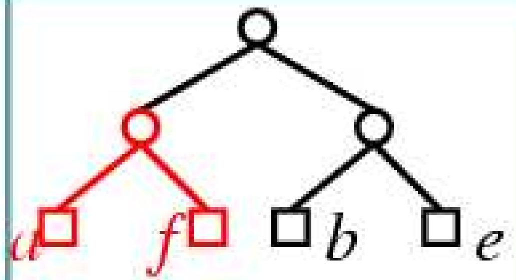
Lazy Updates: Servers update Local Trees only after an address error.



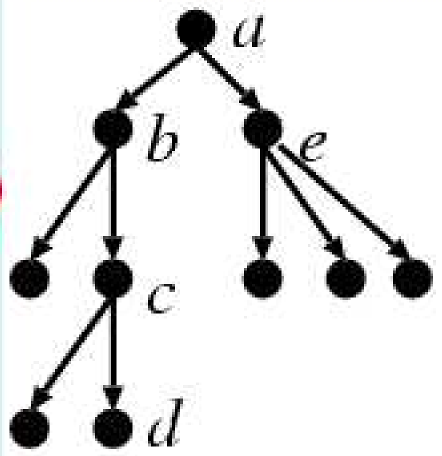
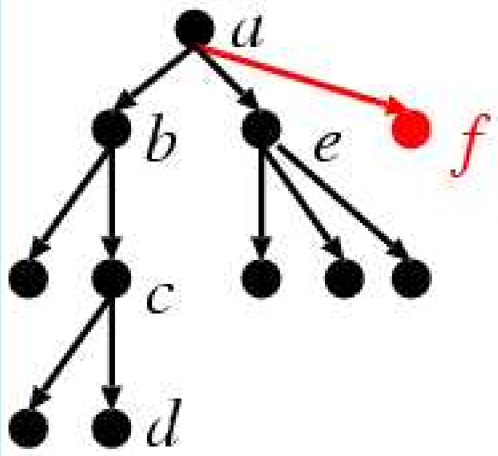
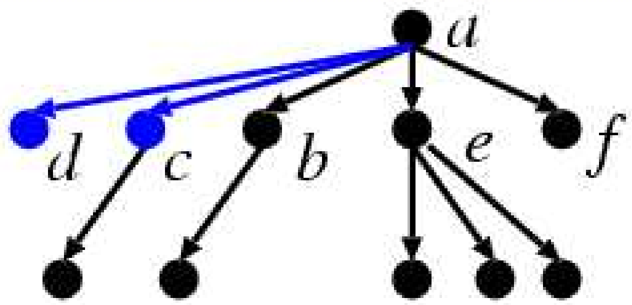
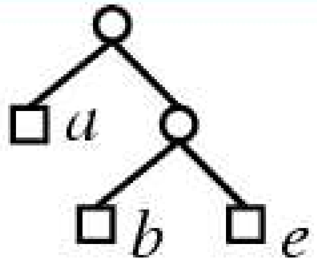
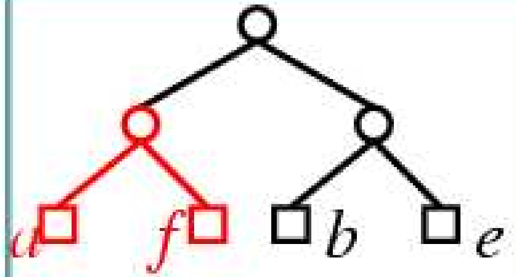
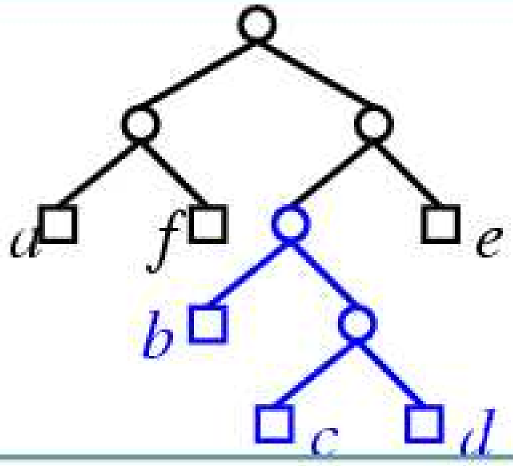
Split Trees

ST(a)	
T(a)	

Split Trees

		Split of a
ST(a)		
T(a)		

Split Trees

		Split of a	*A correction*
ST(a)			
T(a)			

DRT* (Upper Bound 1)

•We analyze the cost of a sequence of requests in a DRT* through splits and searches over the split trees.

•There is a correspondence between DRT* and the Set Union Problem (Di Pasquale, Nardelli: ADBIS 2000, WDAS 2000).

•Idea:To a sequence σ of m requests, producing an n-servers DRT*, we associate a sequence ρ of m finds, n make-sets and $u<n$ unions in SUP.

Amortized Case:

• $O(m \cdot \log_{1+m/n} n)$ messages for a sequence σ of m single-key requests producing an n-server DRT* (Di Pasquale, Nardelli: ADBIS 2000).

Results from Set Union Problem

[Tarjan, Van Leeuwen: 1984]

- Let H be the maximum height of compressed trees in an instance of the Set Union Problem without performing any compressions.

Two important results in the Set Union Problem. Consider a a sequence σ of n make-set and m operation among find and unions. Then:

- If H is $O(n)$ $\Rightarrow$ the cost of σ is $O(m \cdot \log_{1+m/n} n)$.
- If H is $O(\log n)$ $\Rightarrow$ the cost of σ is $O(m \cdot \alpha(m,n))$.

Improvement of DRT* Results

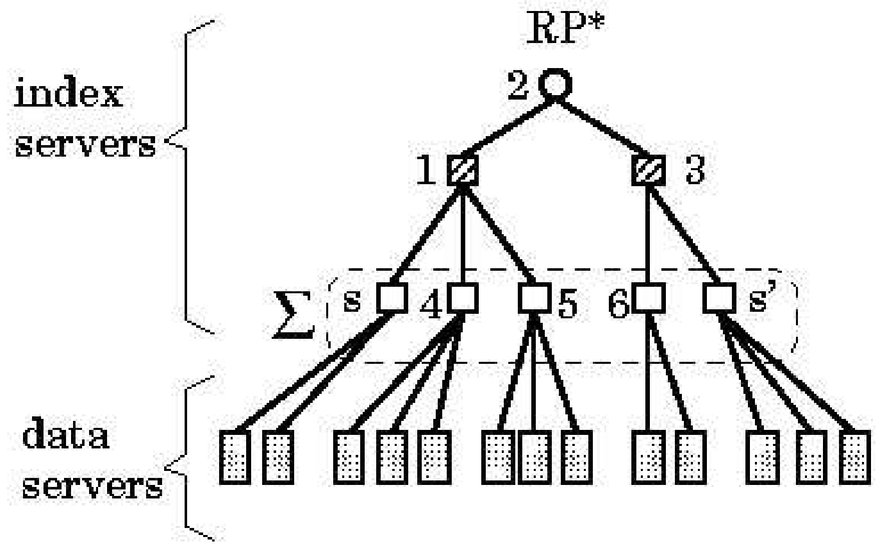
The idea is to perform correction after some splits following a given policy.

Amortized Case Result:
$O(m\ \alpha(m,n))$ messages for a sequence σ of m single-key requests producing an n-servers DRT*. (Di Pasquale, Nardelli, Proietti: WDAS 2002).

DRAWBACK: it doesn't support very well high-concurrency environments.

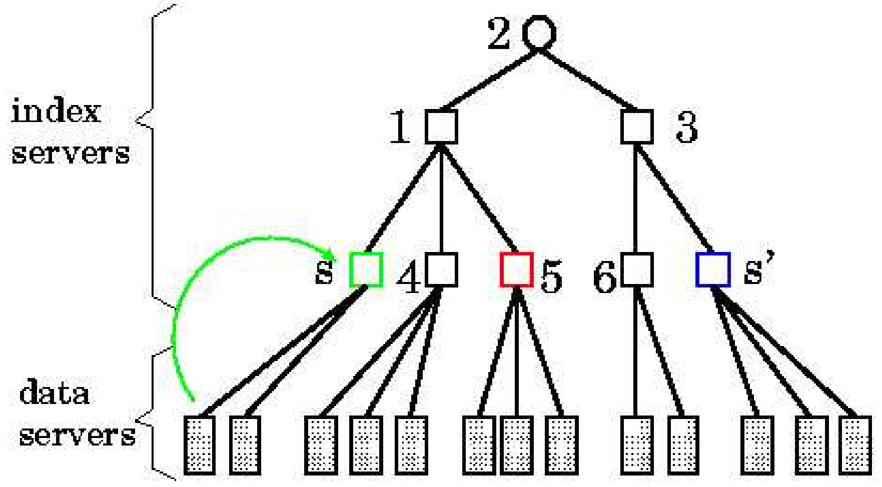
RP* Plus

- We apply the technique to the tree made up by internal nodes (Index Servers).
- For a new server s, the Local Tree of s (LT(*s*)) is equal to the set of child pointers of *s*.



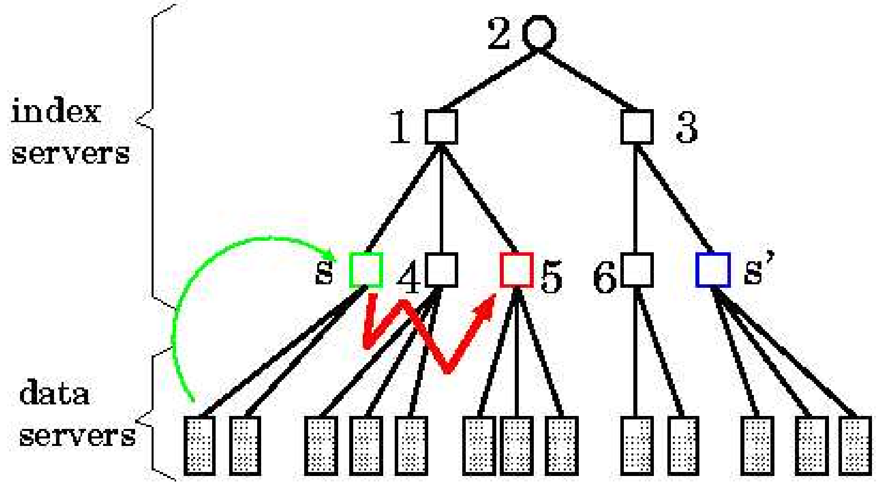
RP* Plus

Data Server receives a request from a Client.
If it is not pertinent for the request, it sends the request to its parent Index Server *s*



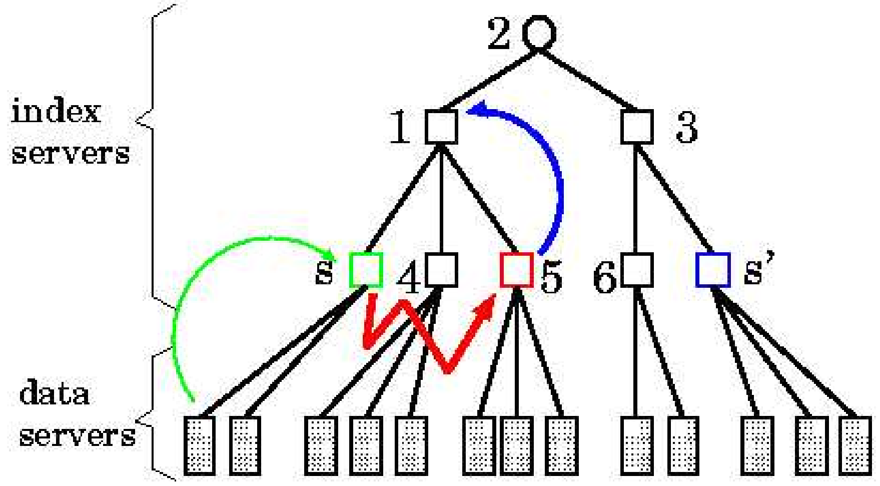
RP* Plus

Index Server s searches in its Local Tree *LT*(*s*) and finds that the pertinent Data Server is one of the sons of Index Server 5. Hence, *s* forwards the request sending a *lt*-message to server 5.



RP* Plus

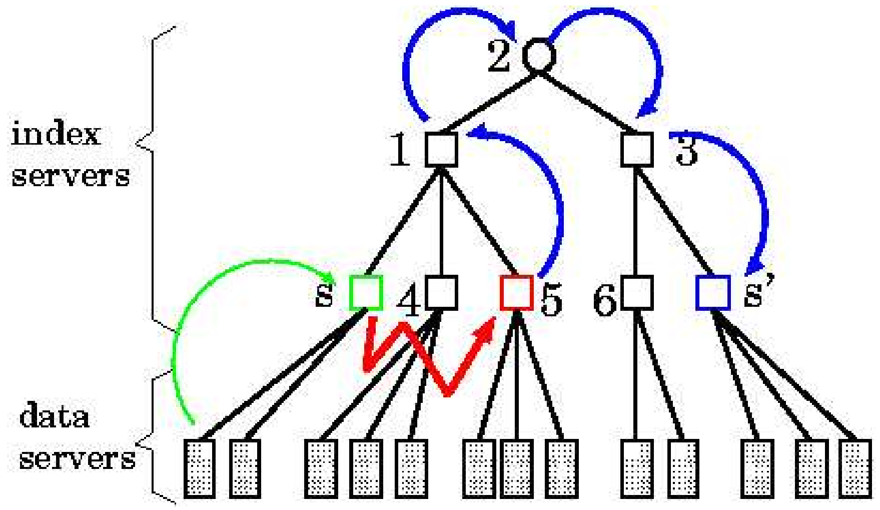
Index Server 5 finds out that it has received a *lt*-message, but it is not the right one, i.e. The request is not pertinent for any son of itself. Hence, it starts a chain of (CT,pp)-messages following the structure of RP* kernel.....



Adriano Di Pasquale
WDAS 2003

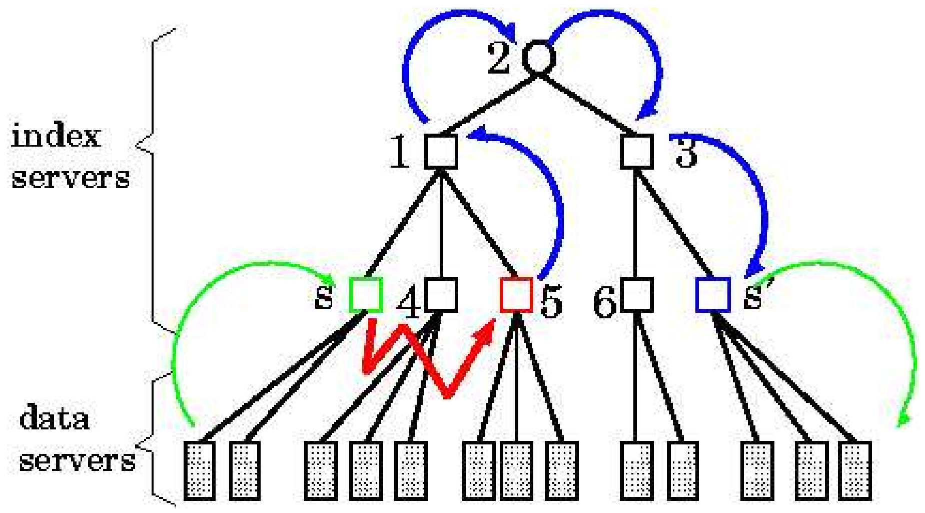
RP* Plus

.... as to arrive to the right Index Server *s'*.



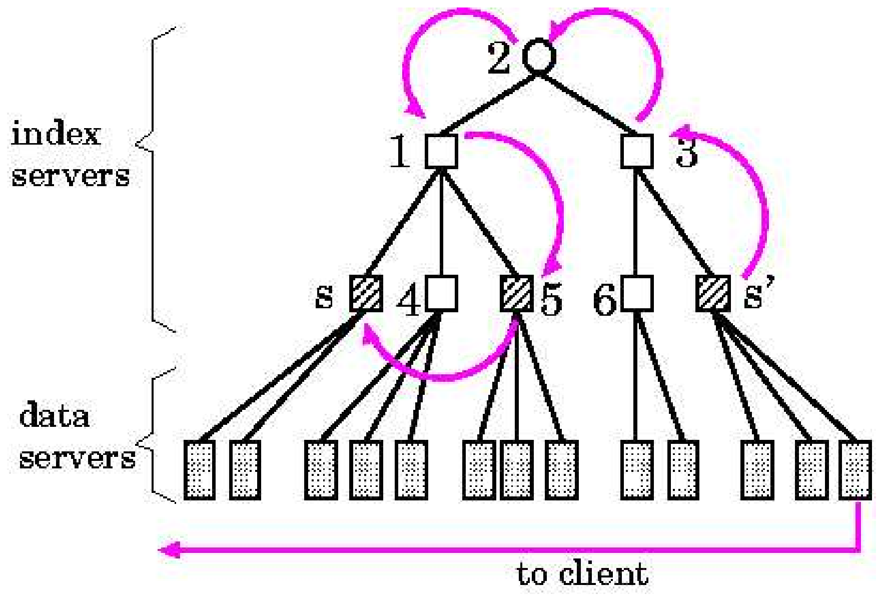
RP* Plus

Finally, Index Server *s'*, sends the request to the pertinent Data Server.



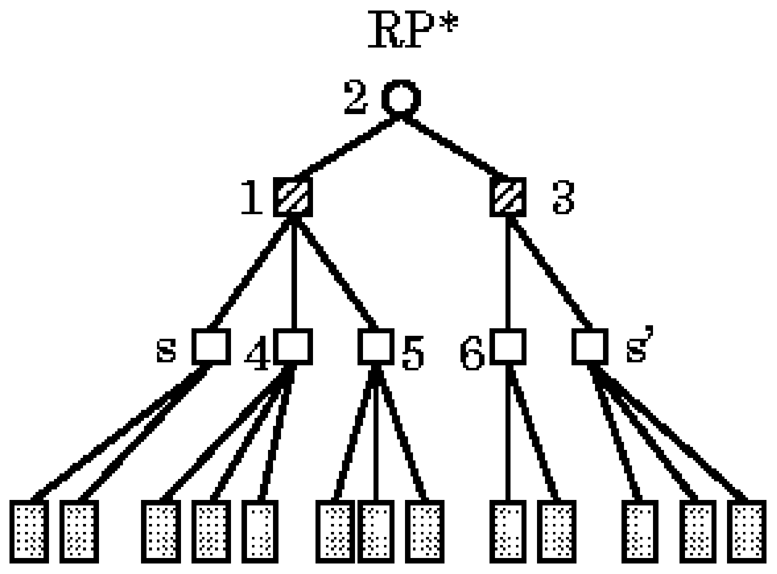
RP* Plus

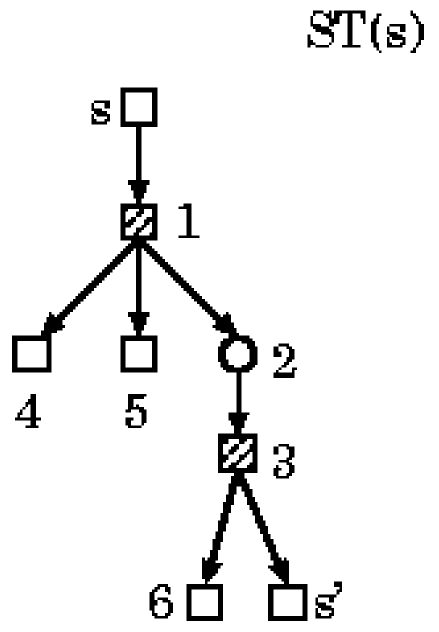
Of course, after that, ICMS are sent to the involved *Index* servers, in order to improve their Local Trees.



Split Tree Variant

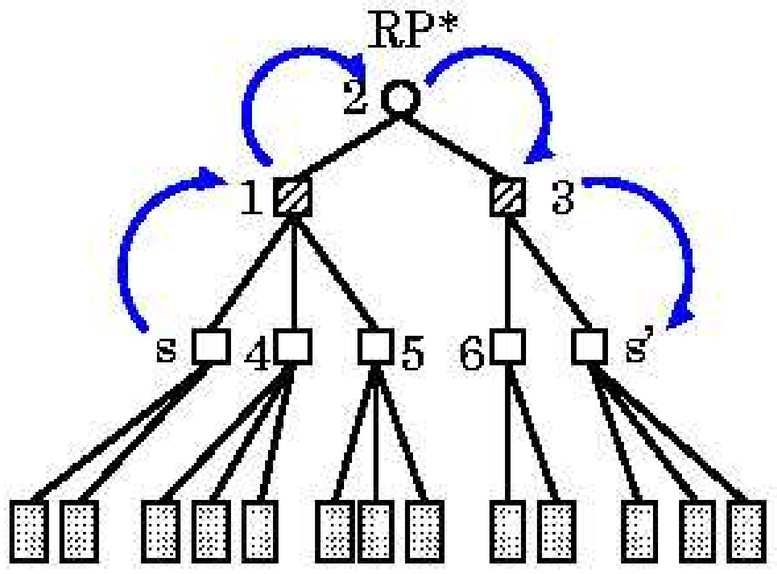
A configuration of Split Tree of server *s* before any address error occurred at itself.

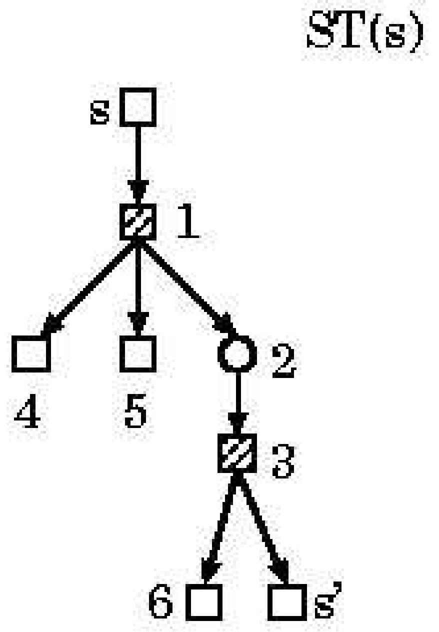




Split Tree Variant

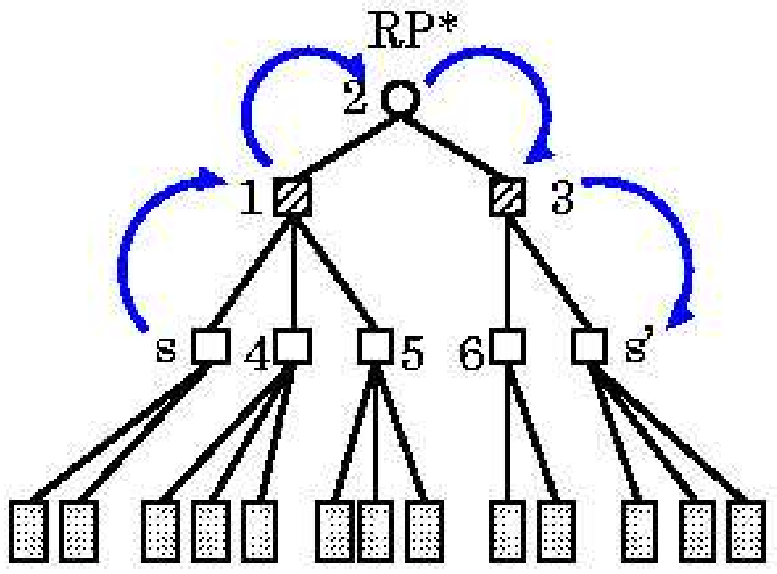
A configuration of Split Tree of server *s* before any address error occurred at itself.

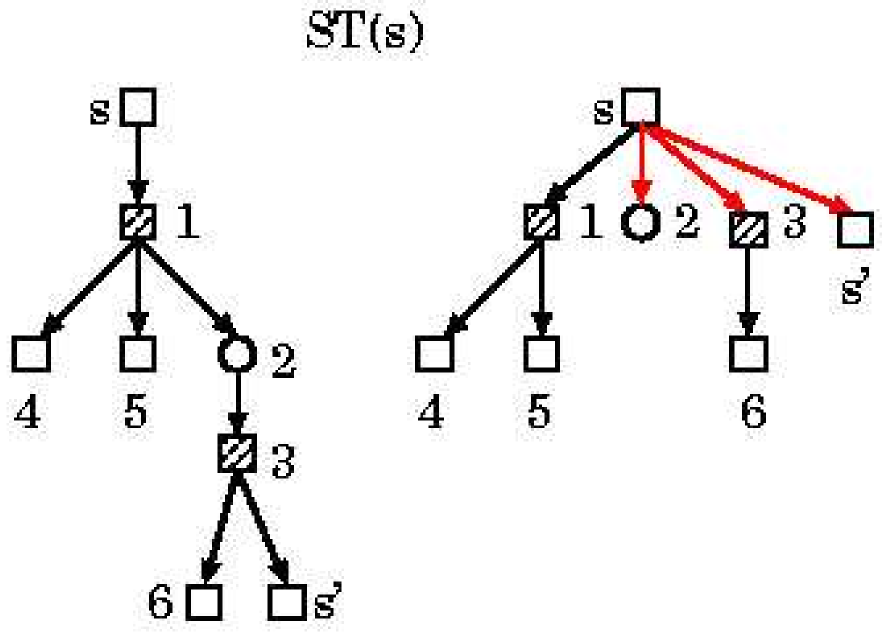




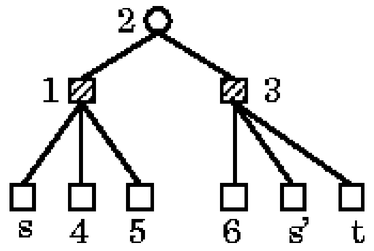
Split Tree Variant

A configuration of Split Tree of server *s* before any address error occurred at itself.



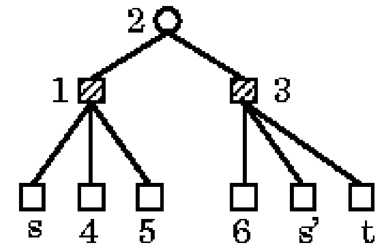


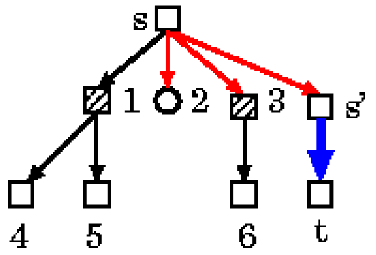
Split Tree Variant

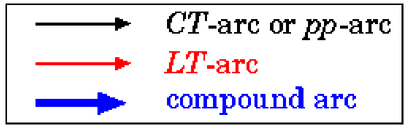


→ *CT*-arc or *pp*-arc
→ *LT*-arc
➡ compound arc

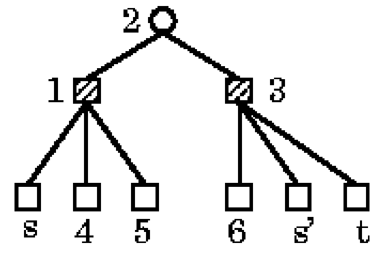
Split Tree Variant

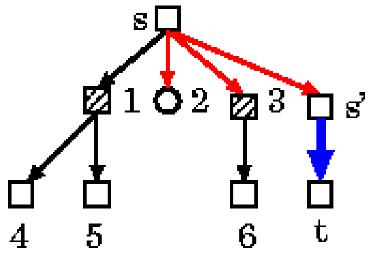


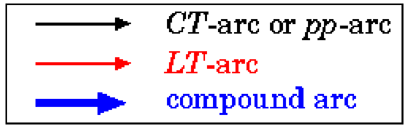


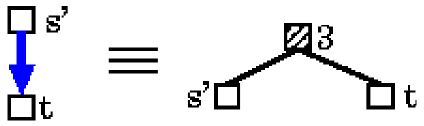


Split Tree Variant

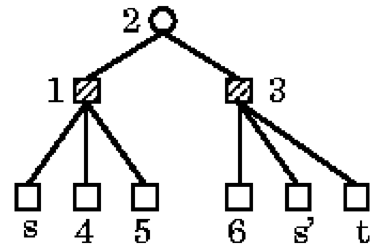


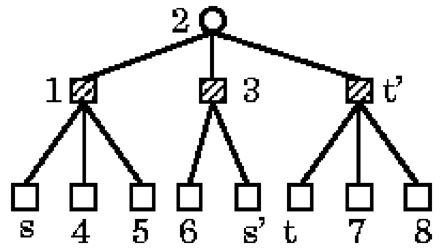


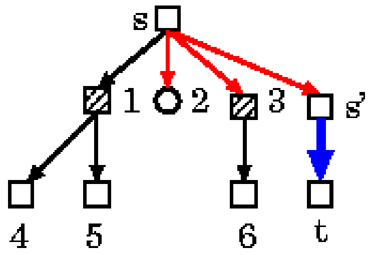


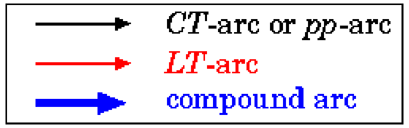


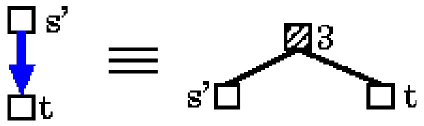
Split Tree Variant



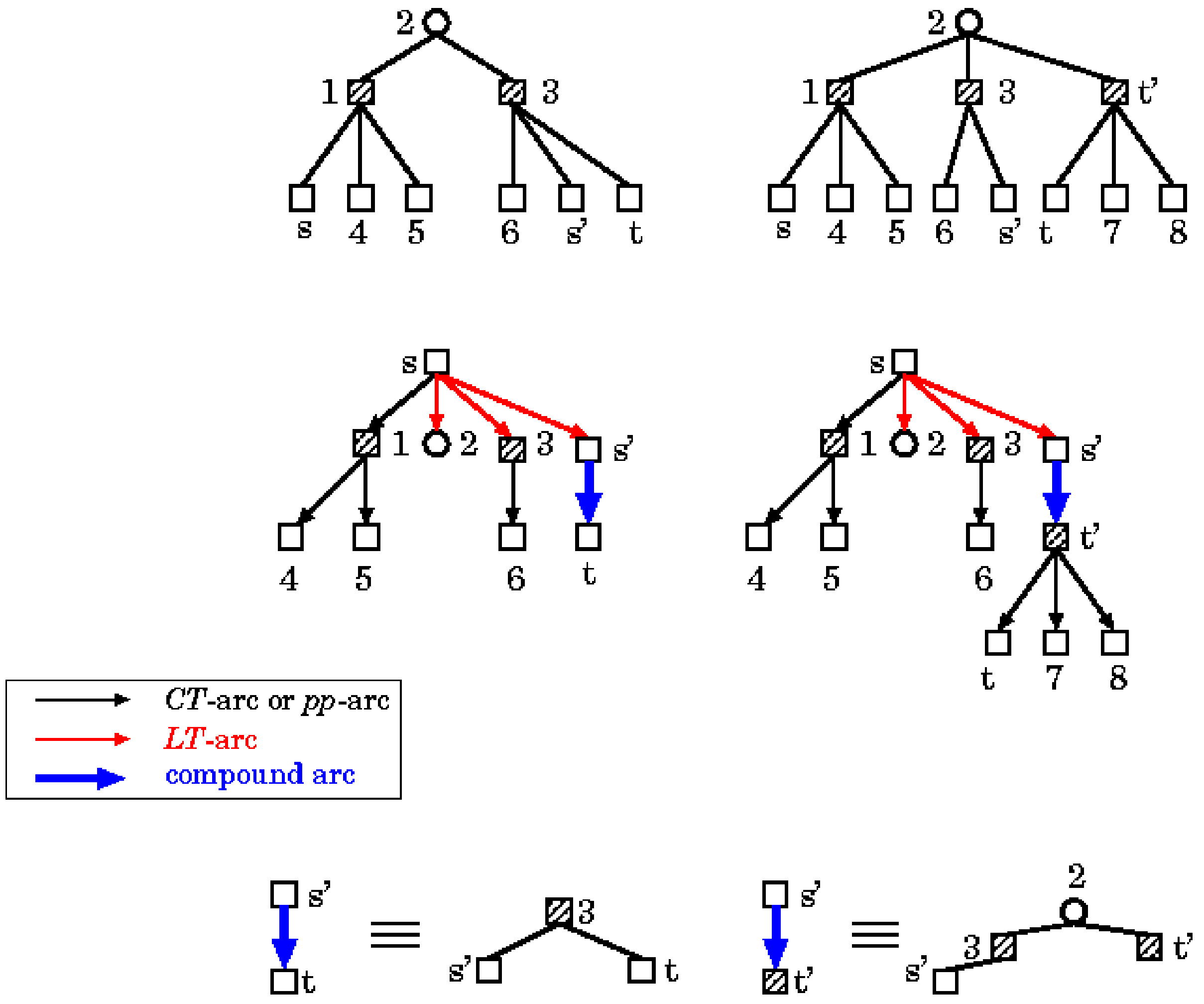




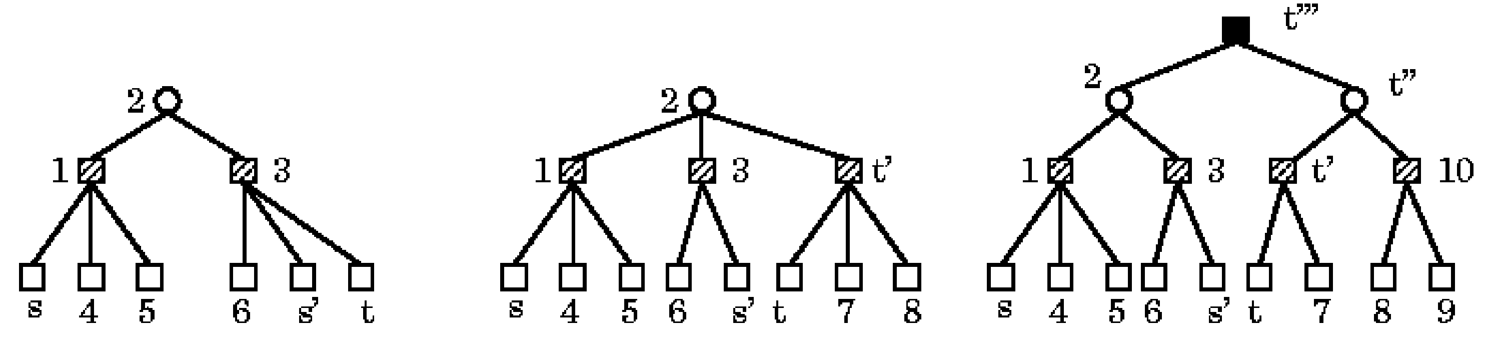


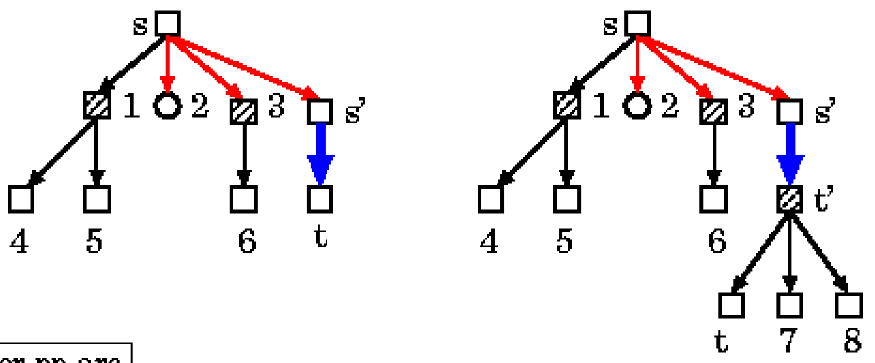


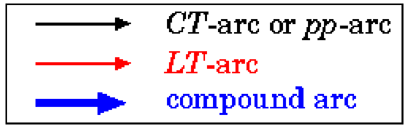
Split Tree Variant

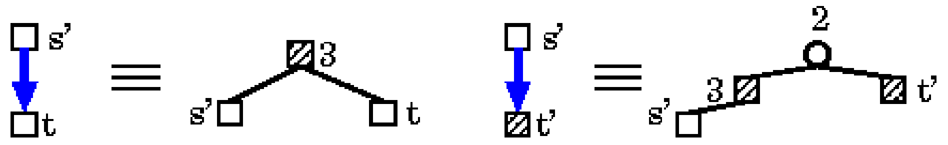


Split Tree Variant

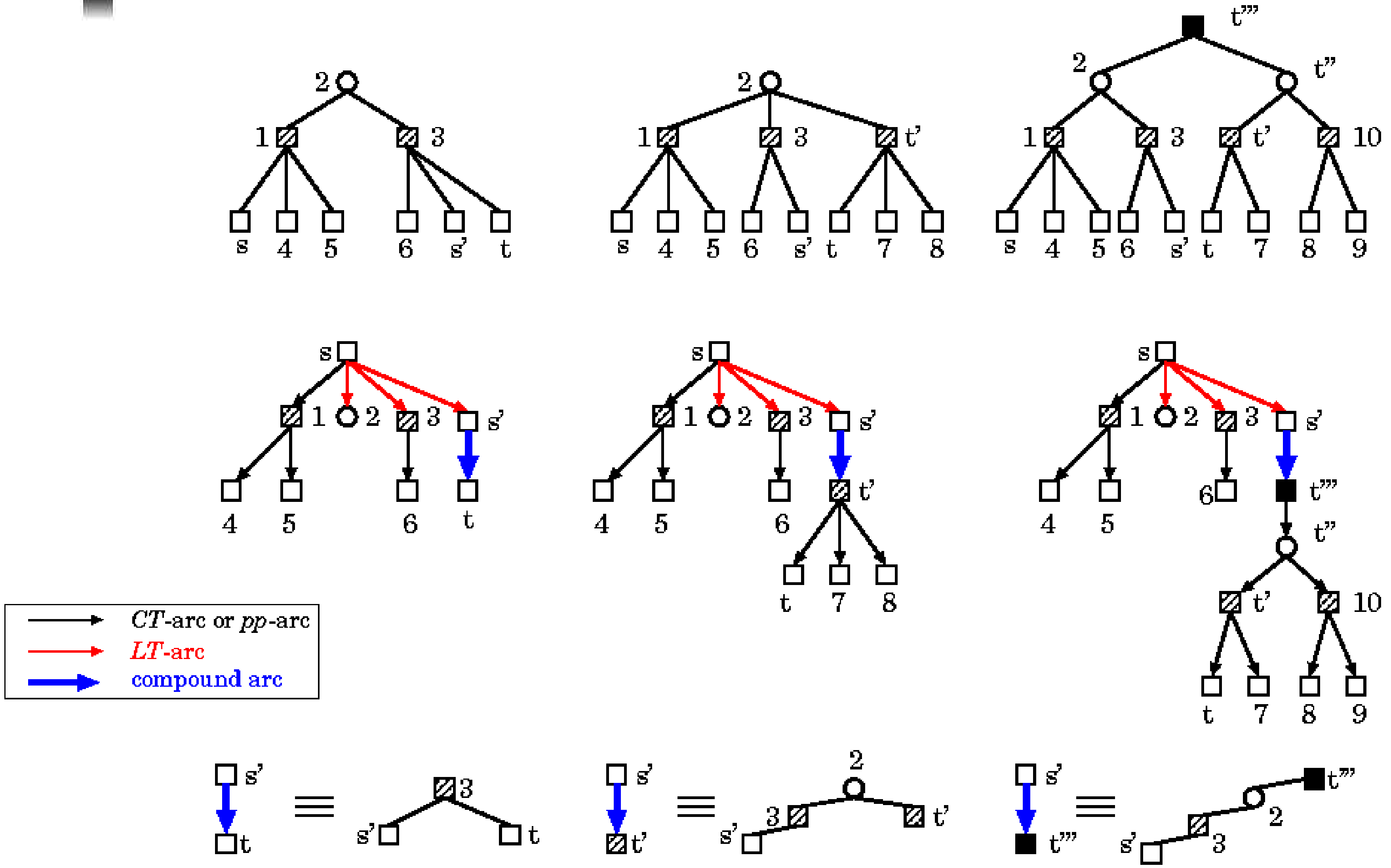




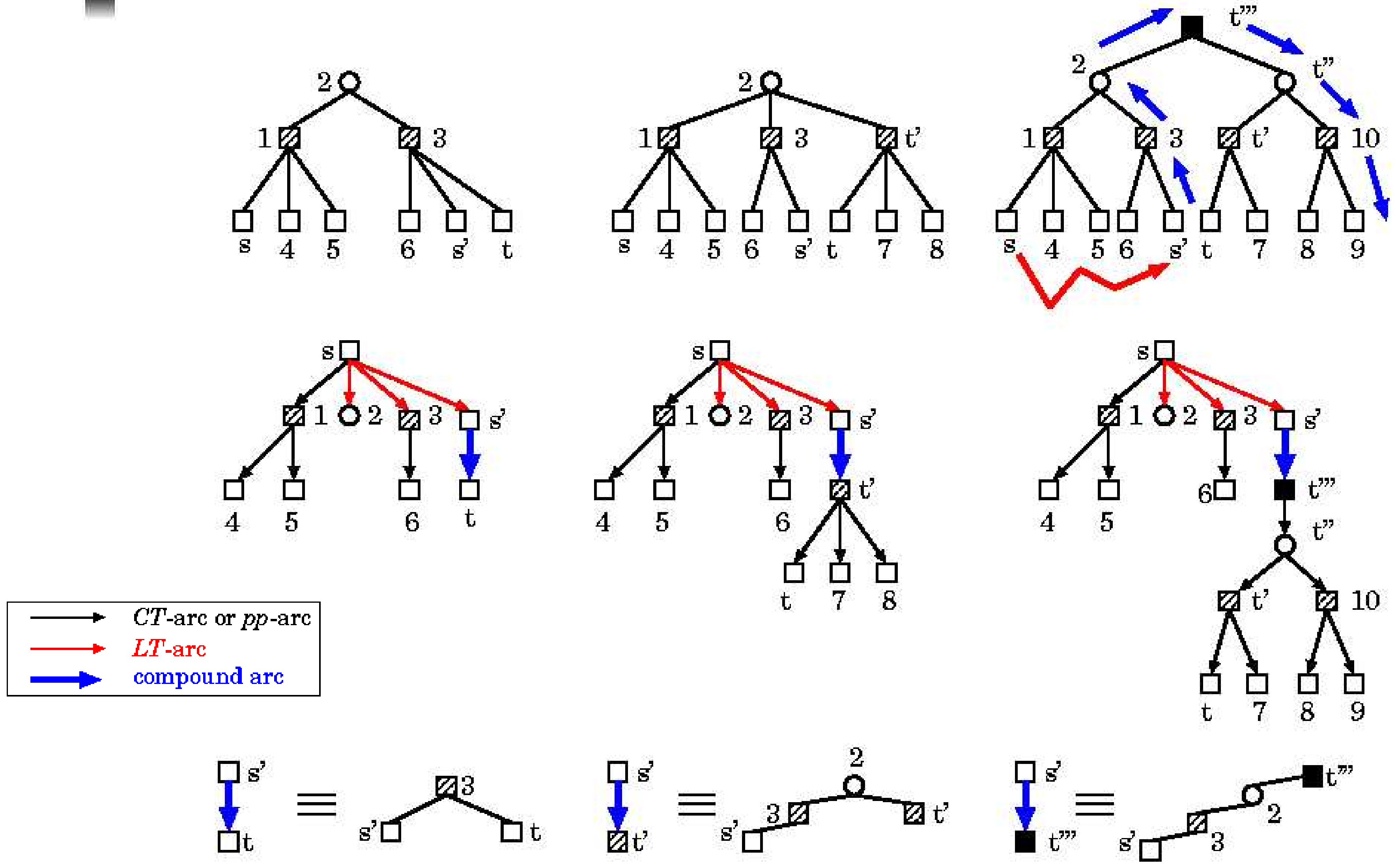




Split Tree Variant



Split Tree Variant



Results

- **Prop1**: Let *s* be the server receiving the request from the client. If *x* server are compressed on *ST*(*s*) at the end, then the request costs *O*(*x*)

- **Prop2**: Let *H* be the maximum height of split trees when we do not perform any corrections after an address error. $H=O(\log n)$, where *n* is the number of Data Servers.

- **Amortized Case Result:**
$O(m\,\alpha(m,n))$ messages for a sequence σ of *m* single-key requests producing an *n*-Data Servers RP*.

High Concurrency

- **DRT*:** In order to bound *H* to *O*(log *n*), and hence to ensure the amortized result, a logarithmic number of servers can be locked in the worst case.

- **RP*:** always a constant (low) number of servers has to be locked.

- Some lock mechanism has to be used to guarantee the amortized result in High Concurrency environments. For this purpose, the variants of the compression technique used in SUP are very useful.

Conclusions and Open Problems

- **RP*s plus:**
  - Amortized almost constant costs.
  - High concurrent data structure.
  - Multi-dimensional management.
  - The strategy can be applied to other variants of search trees.
- **Space scalability:**
  - RP*plus, DRT* and ADST uses *O*(*n*) space for the local index. BDST and RP* uses *O*(1) space for the local index, **but** with logarithmic access costs.
  - What about Order Preserving Hashing Techniques? (Some Idea?)

The End

THANK YOU!!