Just In Time Datastructure

A Scalable Approach to Incremental Data Organization

Saurav Singhi
sauravsi@buffalo.edu
SUNY Buffalo

Darshana Balakrishnan
dbalakri@buffalo.edu
SUNY Buffalo

Oliver Kennedy
okennedy@buffalo.edu
SUNY Buffalo

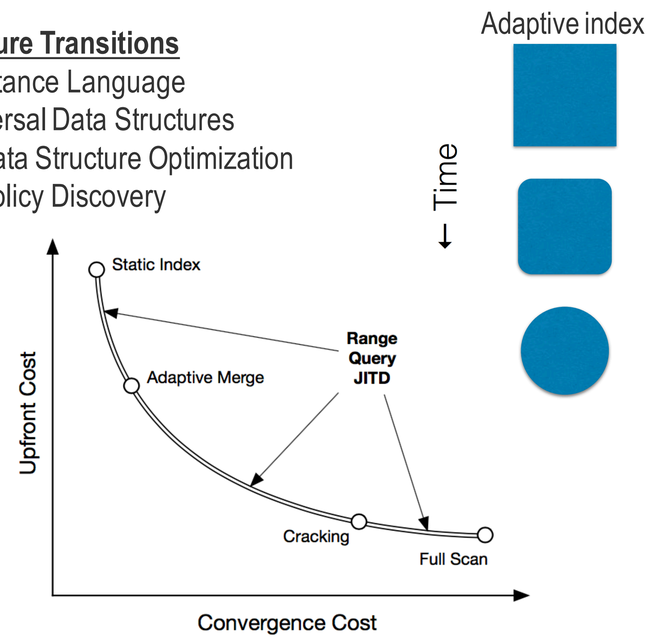
Lukaz Ziarek
lziarek@buffalo.edu
SUNY Buffalo

Background: Adaptive indexing vs Classical indexing

The performance of a Database Management System is closely coupled to the index structures it uses, making index selection an extremely important part of any database deployment. Standard DBMSs often take an all or nothing approach to this problem, where indexes are discarded or built up from scratch, incurring delays during the rebuilding process. As workloads change, index structures must adapt. This gives rise to the need of a structure that can incrementally build indexing based on changing workloads.

Incremental Structure Transitions

1. A Universal Instance Language
2. Realizing Universal Data Structures
3. Just-In-Time Data Structure Optimization
4. Optimization Policy Discovery

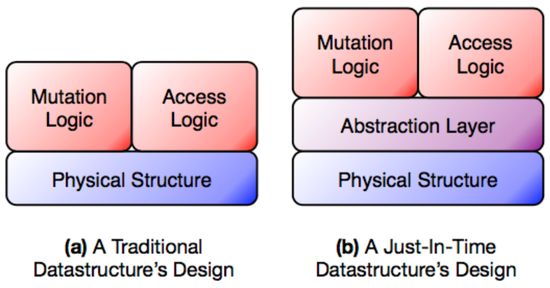


Universal Index

We propose a novel generalization of universal adaptive indexes called just-in-time data structures(JITDs).

Just-in-time data structures:

- Abstraction layer between the physical representation of encoded data
- Logic that accesses and manipulates that physical representation.
- Abstraction helps in designing arbitrary transformations.

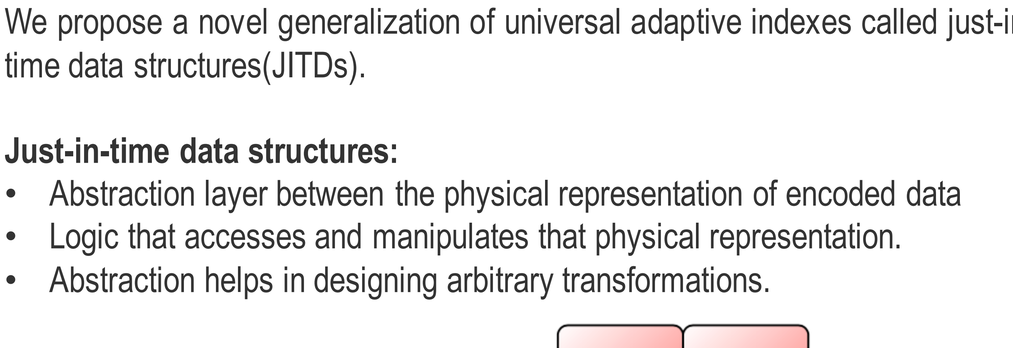


(a) A Traditional Datastructure's Design

(b) A Just-In-Time Datastructure's Design

(a) Primitive Policy: Cracking

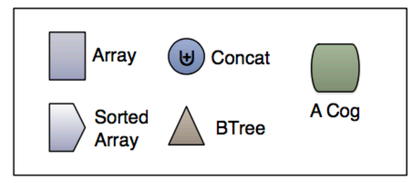
(b) Primitive Policy: Adaptive Merge



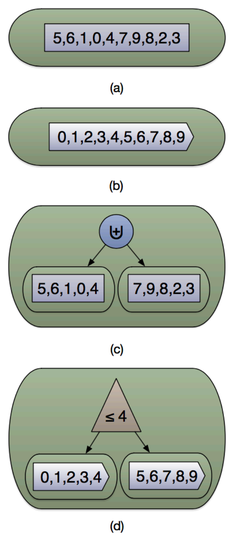
Building Blocks of JITD

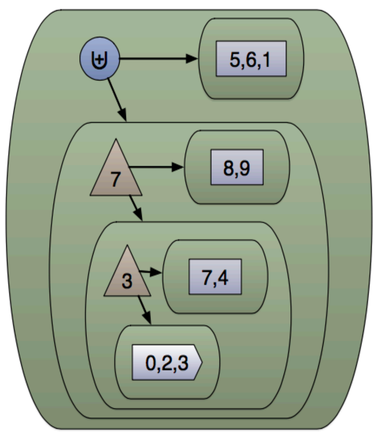
- Cog: Collection of nodes in the data structure irrespective of their type.
- JITDs composed of interchangeable cogs that are consistent in the data they store but need not be consistent in the structure of the nodes.

Nodes	Description
`Array`	An array of N key/value pairs.
`SortedArray`	... in sorted order.
`Concat`	A union of records in 2 child nodes.
`BTree`	... with keys partitioned by separator.



- JITDs can take different node types like Array, Sorted array, Btree depending on Access cost heuristics.
- Decision on which node is to be used is taken by the JITD while reorganizing data.
- In place transformations that restructure the JITD are interleaved with the actual query execution making the JITD an iterative and incremental model rather than a static indexed structure.





Conclusions

- JITDs decouple the logic and physical representation of an index data structure, and allow multiple behaviors, or policies to collectively manipulate a standardized library of physical layout building blocks, or cogs.
- The universal instance language can describe the intermediate state of a datastructure in transition.
- UIL + localized rewrite rules can emulate the behaviors of existing data structures and be hybridized.
- Simulation + Cost-Analysis can be used to derive policies to drive direct rewrites.

References

1. S. Idreos, M. L. Kersten, and S. Manegold. Database cracking. In CIDR, 2007.
2. O.kennedy, L.Ziarek. Just In Time Datastructures. In CIDR, 2015.