



# Using HiCarbString for handling large-volume web documents

---

*Seungbeom Kim <sbkim@dsg.stanford.edu>*

*with*

*David Cheriton <cheriton@dsg.stanford.edu>*

*in*

*Distributed Systems Group, Stanford University*



# Problem

---

- Plethora of textual data
  - The volume is huge
  - Data often contains lots of duplication
    - ex. web archive
- How do we store and handle such data efficiently?
  - Use less space for storage (exploiting duplication)



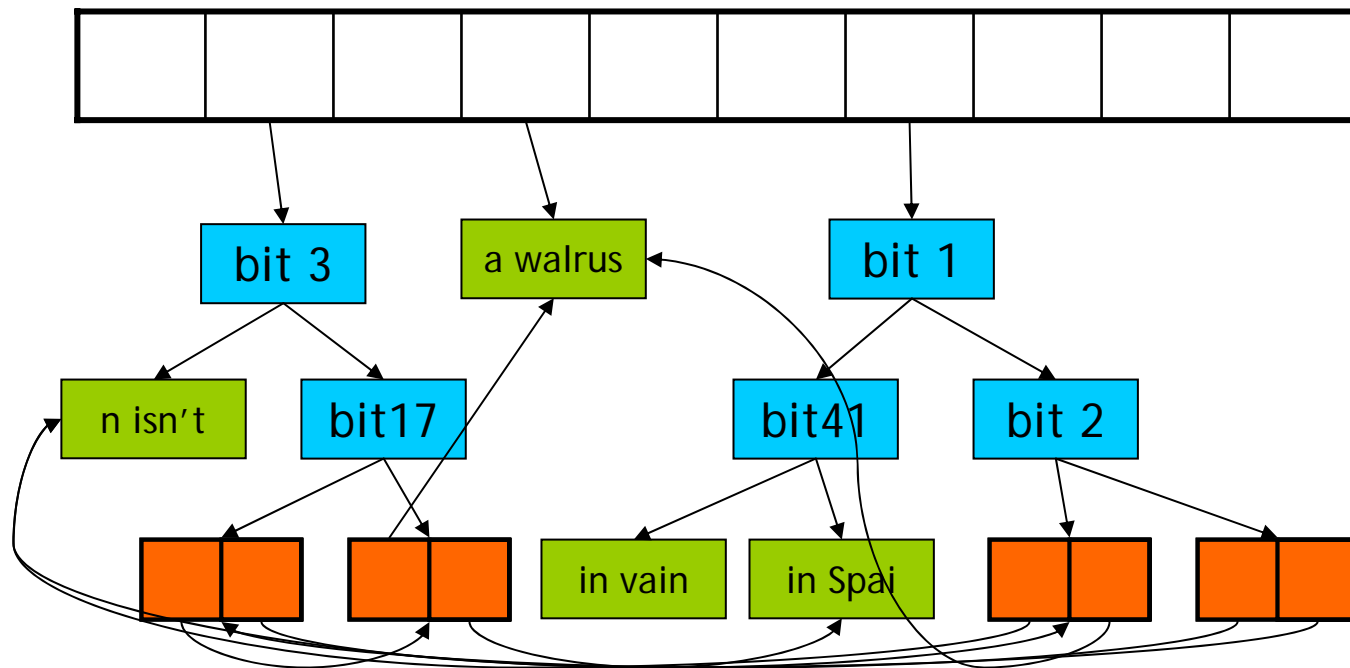
# Solution: HiCarbString

---

- Hierarchical **I**mmutable **C**ontent  
**A**ddressable **R**eferenced-counted **B**locks
- Fixed-size blocks form a Patricia Trie
- All blocks are hashed and stored in a “content directory” that ensures only one block with given data resides in memory

# HiCarbString

- “a walrus in Spain isn’t a walrus in vain”





# HiCarbString

---

- Advantages:
  - Bounded memory usage
  - No external memory fragmentation
  - Sharing of identical values
  - Fast lookup/equality/assignment operations
- Disadvantages:
  - Immutable; modification is expensive
  - Not contiguous; scanning through is more expensive and has lower locality of reference



# Web crawler

---

- Huge volume of string data
- Data often contains lots of duplication  
→ Ideal for HiCarbString!
- Our task:
  - Read an already crawled web data from file (+80,000 documents, 2.0 GB total) and store them in the memory
  - Total space for string data is bounded



# Tokenization

---

- Question: How should we tokenize?
  - By pages? By lines? By words?  
By HTML structure?
  - Options
    - skip single spaces
    - squeeze multiple spaces
    - ...
- Ultimately depends on the application:  
What does it do with the data?



# Tokenization

---

- Trade-off: sharing vs. overhead
  - Smaller tokens: more sharing, more overhead
  - Each string object takes up the space of a pointer (at least).
  - Better not have too many small objects
- Trade-off: sharing vs. manipulability
  - Smaller tokens are less convenient to treat as a long concatenated string



# Space usage

---

- $L$  = sum of lengths of all strings
- $B$  = total space used for StrBlock objects
- $P$  = total space used for string objects (essentially pointers)
- $H = B + P$  = total space used for HiCarbString
  
- $H/L$  = space used / input data size:  
a measure of space overhead



# Space usage

---

- `std::string` also has space overhead
- $S$  = total space used by `std::string` objects
- $S/L$  = a measure of space overhead
- $H/S$  = HiCarbString space usage compared to `std::string` space usage (a fairer comparison)



# Space usage

---

- by word:  
 $H/S = 1.07/3.32 = 0.323$  (67.7% saving)
- by word, with spaces eliminated:  
 $H/S = 0.651/1.86 = 0.350$  (65.0% saving)
- by HTML structure:  
 $H/S = 1.61/1.58 = 1.018$  (1.8% overhead)



# Space usage

---

- $R$  = average ref. count over all StrBlocks:  
a measure of sharing (1=no sharing at all)
- by word:  $R = 23.7$
- by word, with spaces eliminated:  $R = 10.2$
- by HTML structure:  $R \approx 1.2\sim 1.8$



# Runtime

---

- seconds per document,  
compared to `std::string` version:
- by word:  
 $0.0128558 / 0.00935929 = 1.37359$  (+37%)
- by HTML structure:  
 $0.0197144 / 0.0151944 = 1.29748$  (+30%)



# Other applications

---

- *wordcount* (simple word counter)  
takes 0.92x time of `std::string` version
- *webcpp* (formats source code into HTML with syntax highlighting), tokenized  
takes 1.10x time of `std::string` version
- Performance overhead can be made negligible compared to `std::string`



# Conclusion

---

- HiCarbString can reduce the storage space requirement (up to 67%), exploiting duplication in the data.
- The runtime overhead of HiCarbString is not very significant, and can even be negative depending on the application (e.g. with lots of lookup operations).



# Future work

---

- Try more tokenization methods
  - A more versatile tokenizer module
  - Try more combinations of options
  - Recognize bigger common substrs as tokens
- Employ HiCarbString in systems such as *BigTable* and prove its usefulness
- Reduce the space/runtime overhead further