Fast Compaction Algorithms for NoSQL Databases

Mainak Ghosh\textsuperscript{1} and Indranil Gupta\textsuperscript{1} and Shalmoli Gupta\textsuperscript{1} and \textbf{Nirman Kumar}\textsuperscript{2}

\textsuperscript{1}University of Illinois, Urbana-Champaign

\textsuperscript{2}University of California, Santa-Barbara

July 2, 2015
Big Data is Everywhere
Big Data is Everywhere

Systems experts have to cope with Big Data

Online shopping, content management, finance, education
Big Data, NoSQL and Compaction

BIG DATA

NoSQL DB

Write Performance

Read Performance

COMPACTION
Big Data, NoSQL and Compaction

We provide algorithms with provable guarantees for compaction
Results and Recommendation

Use algorithm : Balanced Tree with Smallest Input
A day in the life of a NoSQL DB

Insert, Update, Delete operation

Key: 10, Data: ...

Memtable

Key: 30, Data: ...
Key: 10, Data: ...
Key: 20, Data: ...

SSTable

Key: 10, Data: ...
Key: 20, Data: ...
Key: 30, Data: ...

Before Compaction

SSTable 1
Key: 10, Data: ...
Key: 20, Data: ...
Key: 30, Data: ...

SSTable 2
Key: 20, Data: ...
Key: 30, Data: ...
Key: 40, Data: ...

... SSTable n
Key: 10, Data: ...
Key: 20, Data: ...
Key: 40, Data: ...

After Compaction

SSTable n + 1
Key: 10, Data: ...
Key: 20, Data: ...
Key: 30, Data: ...
Key: 40, Data: ...
Outline

- Compaction - the problem
  - Compaction - our approach
  - Greedy algorithms for Compaction
  - Experimental results
Compaction - so why is it a problem?

**Strategy A**

\[
\begin{align*}
  S_1 & : 1 \\
  S_2 & : 2 \\
  S_3 & : 3 \\
  S_4 & : 1,2,3
\end{align*}
\]
Compaction - so why is it a problem?

**Strategy A**

Cost = $|S_1| + |S_2| + |S_1 \cup S_2| = 4$
Compaction - so why is it a problem?

**Strategy A**

\[
S_1 \cup S_2 \cup S_3 \\
\begin{array}{c}
1,2,3
\end{array}
\]

\[
S_4 \\
\begin{array}{c}
1,2,3
\end{array}
\]

Cost = \(|S_1| + |S_2| + |S_1 \cup S_2| = 4\)

Cost = \(|S_1 \cup S_2| + |S_3| + |S_1 \cup S_2 \cup S_3| = 6\)
Compaction - so why is it a problem?

Strategy A

\[ S_1 \cup S_2 \cup S_3 \cup S_4 \]

Cost = \(|S_1| + |S_2| + |S_1 \cup S_2| = 4\)

Cost = \(|S_1 \cup S_2| + |S_3| + |S_1 \cup S_2 \cup S_3| = 6\)

Cost = \(|S_1 \cup S_2 \cup S_3| + |S_4| + |S_1 \cup S_2 \cup S_3 \cup S_4| = 9\)
Compaction - so why is it a problem?

OR ...
Compaction - so why is it a problem?

Strategy B

\[
\begin{align*}
S_1 & \quad S_2 & \quad S_3 & \quad S_4 \\
1 & \quad 2 & \quad 3 & \quad 1,2,3
\end{align*}
\]
Compaction - so why is it a problem?

**Strategy B**

\[ S_1 \quad \quad S_2 \quad \quad S_3 \cup S_4 \]

\[
\begin{align*}
S_1 & = 1 \\
S_2 & = 2 \\
S_3 \cup S_4 & = 1,2,3
\end{align*}
\]

Cost = |$S_3$| + |$S_4$| + |$S_3 \cup S_4$| = 7
Compaction - so why is it a problem?

Strategy B

\[
\begin{align*}
S_1 & \quad S_2 \cup S_3 \cup S_4 \\
1 & \quad 1,2,3
\end{align*}
\]

\[
\text{Cost} = |S_3| + |S_4| + |S_3 \cup S_4| = 7
\]

\[
\text{Cost} = |S_2| + |S_3 \cup S_4| + |S_2 \cup S_3 \cup S_4| = 7
\]
Compaction - so why is it a problem?

**Strategy B**

\[
S_1 \cup S_2 \cup S_3 \cup S_4
\]

1,2,3

Cost = \(|S_3| + |S_4| + |S_3 \cup S_4| = 7\)

Cost = \(|S_2| + |S_3 \cup S_4| + |S_2 \cup S_3 \cup S_4| = 7\)

Cost = \(|S_1| + |S_2 \cup S_3 \cup S_4| + |S_1 \cup S_2 \cup S_3 \cup S_4| = 7\)
Compaction - so why is it a problem?

Which choice is better?
Current approaches to Compaction

- **Merge sstables when their number exceeds a threshold**
  - Bigtable, Cassandra, Riak
Current approaches to Compaction

- **Merge sstables when their number exceeds a threshold**
  - Bigtable, Cassandra, Riak

- **Merge at every Insert, Update, Delete**
  - Optimizes for reads
  - Cassandra, Riak
Current approaches to Compaction

- **Merge sstables of equal size**
  - Cassandra, Bigtable
Current approaches to Compaction

- **Merge sstables of equal size**
  - Cassandra, Bigtable

- **Prioritize more recent data**
  - Logs
Outline

- Compaction - the problem
- Compaction - our approach
  - Greedy algorithms for Compaction
  - Experimental results
Goals and methods

Theoretical analysis of compaction
Goals and methods

Theoretical analysis of compaction

Formulate as an optimization problem
Study algorithms and their complexity
Theoretical analysis - why?

- Theoretical analysis will provide new insights
Theoretical analysis - why?

- Theoretical analysis will provide new insights

- Better idea about limits of optimization
Theoretical analysis - why?

- Theoretical analysis will provide new insights

- Better idea about limits of optimization

Past work: Mathieu et. al.

- Merge prefix of sets - what prefix?
- Cost function additive
Compaction as an optimization problem

Given sets $S_1, \ldots, S_n$, merge them to produce a single set
Compaction as an optimization problem

A merge is replacing some sets with their union
Compaction as an optimization problem

How many sets $\kappa$?
Compaction as an optimization problem

\[ \text{Cost} = \underbrace{|S_{i_1}| + \ldots + |S_{i_k}|} \quad \text{Read} \quad \underbrace{|S_{i_1} \cup \ldots \cup S_{i_k}|} \quad \text{Write} \]
Compaction as an optimization problem

How to do the merge so that overall cost is minimized?
Compaction as an optimization problem

How to select the sets at each merge?
The tree view

Trees as blueprints for merging algorithms
The tree view

Choice in the merge tree
The tree view

Choice in the ordering of sets on leaves
The compaction problem is NP-hard

Proof in the paper
Hardness result

View - find the tree and the ordering
Fixed tree - choice on ordering. Is this NP hard?
Yes. For some trees we prove NP hardness
Compaction reduces to this problem
Outline

- Compaction - the problem
- Compaction - our approach
- Greedy algorithms for Compaction
- Experimental results
Greedy strategies

Maintain a collection:
$S_1, \ldots, S_n$
Greedy strategies

We always merge 2 sets at a time
Greedy strategies

Strategy determines which two we merge
Greedy strategies

We have $n - 1$ merge steps in all
Greedy strategies

Main intuition: Merge so that larger sets form later
Greedy strategies

Smallest Output (SO)

\[
\text{Minimize } |S_i \cup S_j|
\]
Greedy strategies

Smallest Input (SI)

Choose two smallest sets
Greedy strategies

Balanced Tree (BT)

Ensure a balanced merge tree
Greedy strategies

Largest Match (LM)

Maximize $|S_i \cap S_j|$
Greedy strategies

Balanced Tree, Smallest Input (BT(I))

Combine Balanced Tree with Smallest Input
Approximation guarantee

SI, SO, BT are all $O(\log n)$ approximations
Approximation guarantee

\textbf{BT is } \Omega(\log n) \text{ in worst case}
Outline

- Compaction - the problem
- Compaction - our approach
- Greedy algorithms for Compaction

Experimental results
Experimental Setup

- YCSB generated CRUD operations
Experimental Setup

- YCSB generated CRUD operations
- Illinois Cloud computing testbed
Experimental Setup

- YCSB generated CRUD operations
- Illinois Cloud computing testbed

- Phase I YCSB load/run ops - operation count param
  - Memtable size fixed
  - Different key access modes
Experimental Setup

- YCSB generated CRUD operations

- Illinois Cloud computing testbed

- Phase I YCSB load/run ops - operation count param
  - Memtable size fixed
  - Different key access modes

- Merge sstables in Phase II
Experimental Setup

- YCSB generated CRUD operations
- Illinois Cloud computing testbed

- Phase I YCSB load/run ops - operation count param
  - Memtable size fixed
  - Different key access modes

- Merge sstables in Phase II

- Hyperloglog to estimate set sizes with SO (Flajolet et al.)
**Experimental results**

BT(I) is the most efficient in implementation.
Experimental results

BT(I) is the best cost wise
Experimental results

True cost is modeled accurately by our cost function
Experimental results

Performance within constant factor
Open Questions

SO, SI are better than $O(\log n)$ - proof?
Use algorithm: Balanced Tree with Smallest Input