Two-Level Data Compression using Machine Learning in Time Series Database

Xinyang Yu, Yanqing Peng, Feifei Li,
Sheng Wang, Xiaowei Shen, Huijun Mai, Yue Xie

© 2020 Alibaba Group
Agenda

• Background

• Two level compression framework

• Apply machine learning

• Results
Background
Time series database status

Typical scenarios:
- IoT and Sensor Monitoring
- DevOps Monitoring
- Real-Time Analytics

Increasing significantly in the past few years

From DB-Engines Ranking
Major task

**NASDAQ Composite Index**

- **Top level user view**
  - Query
  - Analyze
  - Predict

- **Bottom level system view**
  - Massive read, write
  - Compression, decompression
  - Downsampling, aggregation etc.

- On <timestamp, value> 8B/8B
A key problem

Digital Economy

IoT era

5G network

…

Explosion of time series data

Apply Compression

Save footprint

Improve transfer latency

Improve process performance

How to compress efficiently?
Existing compression solutions

• Snappy: byte level prediction, RLE
• Gorilla: first apply delta-of-delta on timestamp and xor on value data
• MO: remove bit-level packing for parallel processing
• Sprintz: support predict, bit-packing, RLE and entropy coding
• …

• In general:
  • Support single mode, or a few static modes to compress the data
  • Most use bit-packing good for compression efficiency
Two level compression framework
Fig. 1: Four use cases from real scenario; different use cases have different patterns. Fig 1c shows an example that different periods from data could have different patterns; Fig 1d illustrates the preferred compression scheme at different parts of data.

Three characteristics

Time Correlation  Pattern Diversity  Data Massiveness
Model Formalization: Transform

- Transform stage
  - Map raw data to transformed data that can be easily stored.
  - Capture the pattern of raw data.
  - 6 transform primitives; can be extended to fit other patterns.
  - Example: use delta-of-delta (DOD) on a near-linear time series data

<table>
<thead>
<tr>
<th>Name</th>
<th>Desc.</th>
<th>[v_i - v_{i-1}]</th>
<th>[v_{i-1} - v_i]</th>
<th>[v_i \text{xor} v_{i-1}]</th>
<th>[(v_i - v_{i-1}) - (v_{i-1} - v_{i-2})]</th>
<th>[(v_{i-1} - v_{i-2}) - (v_i - v_{i-1})]</th>
<th>[(v_i - v_{i-1}) \text{xor} (v_{i-1} - v_{i-2})]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delta</td>
<td>(v_i - v_{i-1})</td>
<td>0x00000000</td>
<td>0x1a2b3d4e</td>
<td>0x34567aac</td>
<td>0x4e81b80b</td>
<td>0x68acf581</td>
<td></td>
</tr>
<tr>
<td>RD</td>
<td>(v_{i-1} - v_i)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xor</td>
<td>(v_i \text{xor} v_{i-1})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOD</td>
<td>((v_i - v_{i-1}) - (v_{i-1} - v_{i-2}))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDOD</td>
<td>((v_{i-1} - v_{i-2}) - (v_i - v_{i-1}))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DX</td>
<td>((v_i - v_{i-1}) \text{xor} (v_{i-1} - v_{i-2}))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(v_0\) \[\text{0x00000000}\] \[\text{0x1a2b3d4e}\] \[\text{0x34567aac}\] \[\text{0x4e81b80b}\] \[\text{0x68acf581}\]
\(v_1\) \[\text{0x00000000}\] \[\text{0x1a2b3d4e}\] \[\text{0x00000010}\] \[\text{0x00000001}\] \[\text{0x00000017}\]
\(v_2\) \[\text{0x34567aac}\] \[\text{0x1a2b3d4e}\] \[\text{0x00000010}\] \[\text{0x00000001}\] \[\text{0x00000017}\]
\(v_3\) \[\text{0x4e81b80b}\] \[\text{0x00000010}\] \[\text{0x00000001}\] \[\text{0x00000017}\] \[\text{0x00000017}\]
\(v_4\) \[\text{0x68acf581}\] \[\text{0x00000010}\] \[\text{0x00000001}\] \[\text{0x00000017}\] \[\text{0x00000017}\]
Model Formalization: Differential coding

- Differential coding:
  - Encode a value with less space by eliminating the zero bytes.
  - 3 coding primitives; can also be extended
  - Examples: Compress 5B data into 1B

<table>
<thead>
<tr>
<th>Primitive Name</th>
<th>Parameter</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset</td>
<td>offByteShift</td>
<td>1 byte: 1-bit control bits</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 bytes: 2-bit control bits</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 bytes: 3-bit control bits</td>
<td></td>
</tr>
<tr>
<td>Bitmask</td>
<td>maskByteShift</td>
<td>1 value: (up to 6-bit) bitmask</td>
<td></td>
</tr>
<tr>
<td>Trailing-zero</td>
<td>N/A</td>
<td>2-bit (or 3-bit) trailing-zero control bits</td>
<td></td>
</tr>
</tbody>
</table>
Naïve adaptive compression

• Naïve solution: Try all possible combinations for each data point

• Problem: metadata explosion
  • We have to record the compression scheme for each data point
  • At least 2 Byte metadata (primitive choice + primitive parameter) for each data point
  • 25%+ overhead
Observation

- For most time series data,
  - The total number of different patterns is limited
  - Patterns can remain stable for a contiguous time range

Fig. 1: Four use cases from real scenario; different use cases have different patterns. Fig 1c shows an example that different periods from data could have different patterns; Fig 1d illustrates the preferred compression scheme at different parts of data.
Model Formalization: parameterized scheme space

All time series

Parameterized scheme space

<p>|
| --- |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bits</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>majorMode</td>
<td>2</td>
<td>[0, 3]</td>
<td>Indicate the selected major mode</td>
</tr>
<tr>
<td>transType1</td>
<td>3</td>
<td>[0, 5]</td>
<td>Refer to a transform in Table III</td>
</tr>
<tr>
<td>transType2</td>
<td>3</td>
<td>[0, 5]</td>
<td>Similar to transType1</td>
</tr>
<tr>
<td>transType3</td>
<td>3</td>
<td>[0, 5]</td>
<td>Similar to transType1</td>
</tr>
<tr>
<td>offByteShift1</td>
<td>3</td>
<td>[0, 7]</td>
<td>Byte offset for 1-byte offset coding: offByteShift = offByteShift1</td>
</tr>
<tr>
<td>offByteShift2</td>
<td>1</td>
<td>[0, 1]</td>
<td>Byte offset for 2-byte offset coding: offByteShift = offByteShift1 - offByteShift2</td>
</tr>
<tr>
<td>offByteShift3</td>
<td>1</td>
<td>[0, 1]</td>
<td>Byte offset for 3-byte offset coding: offByteShift = offByteShift1 - offByteShift2 - offByteShift3</td>
</tr>
<tr>
<td>offUseSign</td>
<td>1</td>
<td>[0, 1]</td>
<td>Indicate if offset coding use sign</td>
</tr>
<tr>
<td>maskByteShift</td>
<td>3</td>
<td>[0, 5]</td>
<td>Byte offset for bitmask coding</td>
</tr>
</tbody>
</table>

Per timeline

Scheme space

Per point

Compression scheme

Compression scheme

...  

\[ S = \{ s_1, s_2, \ldots, s_n \} \]

- If supports 4 adaptive schemes, only 2 bits, 3.125% metadata overhead

\[ s = (a, b, \lambda_a, \lambda_b) \]

\[ a \in P_{trans}, b \in P_{code}, \lambda_a, \lambda_b \]

are parameters associated to a, b respectively
Solution: 2 level compression framework

**Top Level**
- Extract global characteristic

**Bottom Level**
- Look into each point

**Candidate scheme space**
- Transform preferred
- Offset preferred
- Bitmask preferred
- Mixed preferred

**Major mode selection**
- Decide control parameters for the selected major mode

**Figure out a proper scheme space**

**Find best scheme to compress**

Fig. 2: The two-level compression framework AMMIMO for compressing time-series data.
Rule-based scheme space selection

**Algorithm 4: Rule-based Scheme Space Selection**

- **Input:** metric value sequence $ms$
- **Output:** control parameter values $params$ in Table IV

/* PART I: calculate the benefit_score */
/* benefit_score: the total number of bytes can be saved against the worst case 9-byte original representation per point (i.e., 1 byte of control bits and 8 bytes of differential value) for different compress schemes in a timeline */

```
1 a 6 x 6 array with zero initializations: benefit_score;
2 for each point $ms[i]$ where $i \in [1, ms.length]$ do
3    for each transform_mode $tm[j]$ in Table III do
4       for each coding format $cf[k]$ in Table II do
5          benefit = calculateBenefit($ms[i], tm[j], cf[k]$);
6          benefit_score[$j$][$k$] += benefit;
7 /* PART II: calculate the params based on benefit_score */
/* best_majorMode_score represents the best score among 4 major modes */
8 best_majorMode_score = 0;
9 for each majorMode $mm[i]$ where $i \in [0, mm.length]$ do
10    majorMode_score = 0;
11    /* best_subMode_score represents the best score among 4 sub modes of the majorMode $mm[i]$ */
12    best_subMode_score = 0;
13 for each subMode $sm[j]$ where $j \in [0, sm.length]$ do
14    /* find the array indexes in benefit_score that match $mm[i]$ and $sm[j]$, say $s$ and $t$ */
15    s, t = findIndex($mm[i], sm[j]$);
16    if benefit_score[$s$][$t$] > best_subMode_score then
17        best_subMode_score = benefit_score[$s$][$t$];
18    majorMode_score += best_subMode_score;
19    if majorMode_score > best_majorMode_score then
20        params = findParams($mm[i]$);
```

- **Problem:**
  - Metric maybe not ideal
  - Human manually designed code
  - Not an automatic and adaptive method
Apply machine learning
Use deep reinforcement learning

- Why machine learning?
  - Why reinforcement learning?
    - Not easy to create sample with label
      - 256^256 samples in theory, each need traverse 331,776 choices to figure out best
    - Not ideal one-class-label
    - Should be automatic

Time-series Data

Multi-label classification problem
• 32 points are taken as a basic block
• Duplicate and batch blocks to train, sample options, calculate loss and then do back propagation

\[
\frac{1}{M \times N} \sum_{i=1}^{M \times N} \left( f_n(cop_i) \ast Hcs(cop_i) \right) - \lambda \ast H(cop)
\]  

© 2020 Alibaba Group
Results
Experiment setup

**TABLE VII: Datasets with 28 selected time series.**

<table>
<thead>
<tr>
<th>Test Set A</th>
<th>Name</th>
<th>Points</th>
<th>Name</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>IoT0</td>
<td>430,737</td>
<td>IoT6</td>
<td>430,413</td>
<td>Server35</td>
</tr>
<tr>
<td>IoT1</td>
<td>429,745</td>
<td>IoT7</td>
<td>313,539</td>
<td>Server41</td>
</tr>
<tr>
<td>IoT2</td>
<td>428,390</td>
<td>Server30</td>
<td>158,188</td>
<td>Server43</td>
</tr>
<tr>
<td>IoT3</td>
<td>344,581</td>
<td>Server31</td>
<td>147,385</td>
<td>Server46</td>
</tr>
<tr>
<td>IoT4</td>
<td>306,736</td>
<td>Server32</td>
<td>165,395</td>
<td>Server47</td>
</tr>
<tr>
<td>IoT5</td>
<td>372,868</td>
<td>Server34</td>
<td>140,194</td>
<td>Server48</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Set B</th>
<th>Name</th>
<th>Points</th>
<th>Name</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server57</td>
<td>26,779</td>
<td>Server94</td>
<td>140,198</td>
<td></td>
</tr>
<tr>
<td>Server62</td>
<td>32,569</td>
<td>Server97</td>
<td>158,194</td>
<td></td>
</tr>
<tr>
<td>Server66</td>
<td>135,409</td>
<td>Server106</td>
<td>136,478</td>
<td></td>
</tr>
<tr>
<td>Server77</td>
<td>136,598</td>
<td>Server109</td>
<td>153,438</td>
<td></td>
</tr>
<tr>
<td>Server82</td>
<td>143,798</td>
<td>Server115</td>
<td>165,384</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE VIII: 8 longest time series datasets in UCR**

<table>
<thead>
<tr>
<th>Test Set A</th>
<th>Name</th>
<th>Points</th>
<th>Test Set B</th>
<th>Name</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandOutlines</td>
<td>641,796</td>
<td></td>
<td>CinC_ECG_torso</td>
<td>8,190</td>
<td></td>
</tr>
<tr>
<td>Haptics</td>
<td>19,638</td>
<td></td>
<td>InlineSkate</td>
<td>16,929</td>
<td></td>
</tr>
<tr>
<td>StarLightCurves</td>
<td>155,496</td>
<td></td>
<td>MALLAT</td>
<td>6,138</td>
<td></td>
</tr>
<tr>
<td>UWaveGestureLibraryAll</td>
<td>115,168</td>
<td></td>
<td>Phoneme</td>
<td>4,092</td>
<td></td>
</tr>
</tbody>
</table>

- **Gorilla**: a state-of-the-art commercial bit-level compression algorithm applied in server side
- **MO (Middle-Out)**: a public byte-level compression algorithm good for parallel processing
- **Snappy**: a general-purpose compression algorithm developed by Google
- **AMMMO variants**
  - AMMMO Lazy
  - AMMMO Rnd1000Avg
  - AMMMO Analyze
  - AMMMO ML
  - …
AMMMO performance comparison

- Compression ratio:
  - Snappy << Gorilla/MO << AMMMO
- AMMMO Compression efficiency:
  - GB/s level in GPU platform
ML performance

- ML performs well in compression ratio view
- ML selects similar meaningful parameter value

**TABLE IX: Control settings selected by different AMMMO variants.**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>IoT1</th>
<th>IoT2</th>
<th>Server35</th>
<th>Server48</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>major Mode</td>
<td>trans Type1</td>
<td>trans Type2</td>
<td>offByte Shift1</td>
</tr>
<tr>
<td>Analyze</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>RandomBest</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ML</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>RandomBest</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>ML</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Analyze</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>RandomBest</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>ML</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>RandomBest</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>ML</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Analyze</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>RandomBest</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>ML</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 10: Metric value compression ratios from different methods. Right vertical axis is for Server97 and left axis for other datasets.
Conclusion

• Proposed a two level framework for time-series data compression
  • In detail, we present AMMMO definition, the result shows it achieves ~50% better compression efficiency, and fits parallel computing well

• Designed DRL logic to do scheme space selection (for final compression), which is an automatic, intelligent, and efficient way
Reference
References


References


Thanks!

Q&A