# Main contents in DiagOOM

## 1. Empirical study:

OOM Cases:

|  |  |
| --- | --- |
| Category | Cases |
| Studied | 40 |
| The cause patterns are known | 31 |
| Reproduced | 21 |

\* Reproduced means we run the same/similar job as the one posted on the Web. The thrown OOM stack trace of each job is as same as that posted on the Web.

Cause patterns:

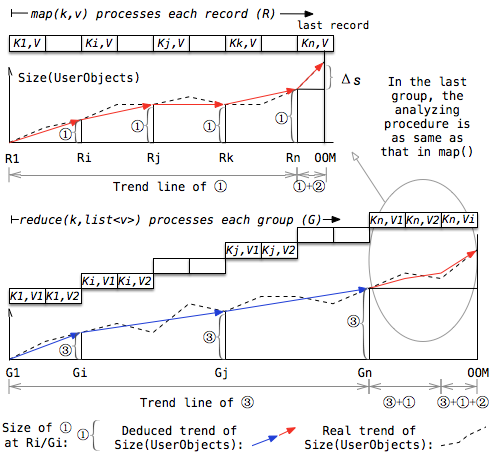
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Configuration | | Hotspot key | User code | | |
| Large framework buffer | Improper data partition | Hotspot key | Large intermediate results | Large accumulated results | Large external data |
| Stackoverflow | 4 (3) | 2 (1) | 2 (1) | 3 (2) | 9 (6) | 6 (4) |
| Hadoop  mailing list | 1 (1) |  |  |  |  |  |
| Developer’s blog |  |  | 1 (1) |  |  |  |
| MapReduce book |  |  | 1 (1) | 1 (0) | 1 (1) |  |
| Total | 5 (4) | 2 (1) | 4 (3) | 4 (2) | 10 (7) | 6 (4) |

Interpretation of the cause patterns:

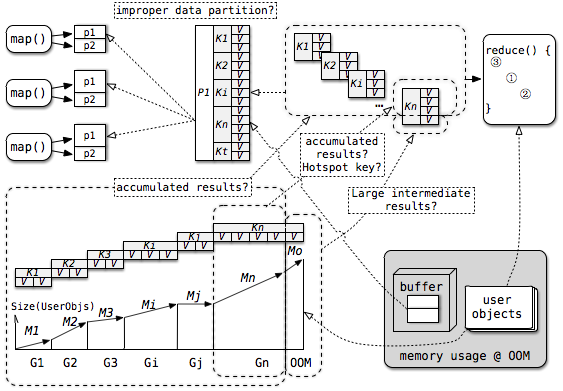
|  |  |  |  |
| --- | --- | --- | --- |
|  | Meaning | Lead to | How to get the patterns |
| Large framework buffer | 1. Large fixed buffer such as byte[] 2. Large virtual buffer (e.g., 70% of heap *can* be used to store the intermediate data) | 1. occupy large memory space 2. more intermediate data are stored in memory | Lowering the buffer size can avoid the error |
| Improper data partition | 1. each partition is large 2. some partitions are much larger than others | 1. more in-memory intermediate data 2. more memory consumption of the user code | Add reduce number or adjust partition function can avoid the error |
| Hotspot key | One or some <K, list(V)> groups are very large | more memory consumption while reduce()/combine() processes these groups | Experts mentioned “hotspot key” or we found some groups are much larger than others |
| Large intermediate results | Large computing results are generated while *a* record is under processing   1. The record is large 2. Even small record can generate large results | High memory consumption while user code processes this record | Users report the input record is very large or we found the memory usage rise dramatically while current record is under processing |
| Large accumulated results | Intermediate results are constantly accumulated in memory | More records are processed, higher the memory consumption | We found user code uses data structure to cache the input records or intermediate results |
| Large external data | No records have been processed but large external data are cached in memory | The external data occupy large memory space | We found that user code reads large data from distributed files before processing the records |

## 2. Cause identification:

Step 1: Use gradient analysis to figure out the trend of memory usage in user code.



Step 2: Use rules to identify the causes patterns of user code and cause-related input records.



|  |  |  |  |
| --- | --- | --- | --- |
| Rules | Cause pattern | Cause-related input records | Next action |
| map():  \*linear trend in [K1, Kn) | Large map-level accumulated results | All the input records | The input data size is the cause-related configuration |
| reduce():  linear trend in [G1, Gn] | Large reduce-level accumulated results | All the input records | check data partition |
| reduce():  linear trend in Gn | Large group-level accumulated results | Records in the last group | check hotspot key |
| \*Sharp growth in the last record Kn | Large intermediate results | The last record  <Kn, Vn> | check hotspot key |
| Other trends | Users need to understand the trend and figure out the cause patterns themselves | Unknown | None |

\*linear: In statistics, pearson correlation is a measure of the linear correlation (dependence) between two variables X and Y, giving a value between +1 and −1 inclusive, where 1 is total positive correlation, 0 is no correlation, and −1 is total negative correlation. Here, we use “pearson(input records, size(user objects)) > 0.90” to denote the linear relationship.

\*Sharp growth in Kn: We will rerun the user code and let it only process the last record Kn. During the processing, we will record the memory growth as “memory growth in Kn”. If the memory growth in Kn is an **outlier** compared with the growth in the previous input records, we regard the growth in Kn as a *sharp* growth.

\*Outlier: In statistics, Q3 + 1.5IQR is the typical rule of thumb to identify outliers in the upper tail. See <http://www.itl.nist.gov/div898/handbook/prc/section1/prc16.htm> for more details. Here, we use the slope of the memory growth in Ki/Gi as the sample. If the memory growth exceeds Q3 + 1.5QR, the growth is an outlier.

一个基准是用BoxPlot来决定适度离群值（mild Outliers）和极限离群值（extreme Outliers），适度离群值是任何值1.5倍大于基于剩下所有的值的IQR，极限离群值是任何值3倍大于剩下所有的值的IQR，IQR（Interquartile Range）代表四分位数间距，是这些值中的50%中间值，分别是Q1-25%, Median-50%,Q3-75%, IQR=Q3-Q1

Step 3: Identify the other cause patterns and the cause-related configurations.

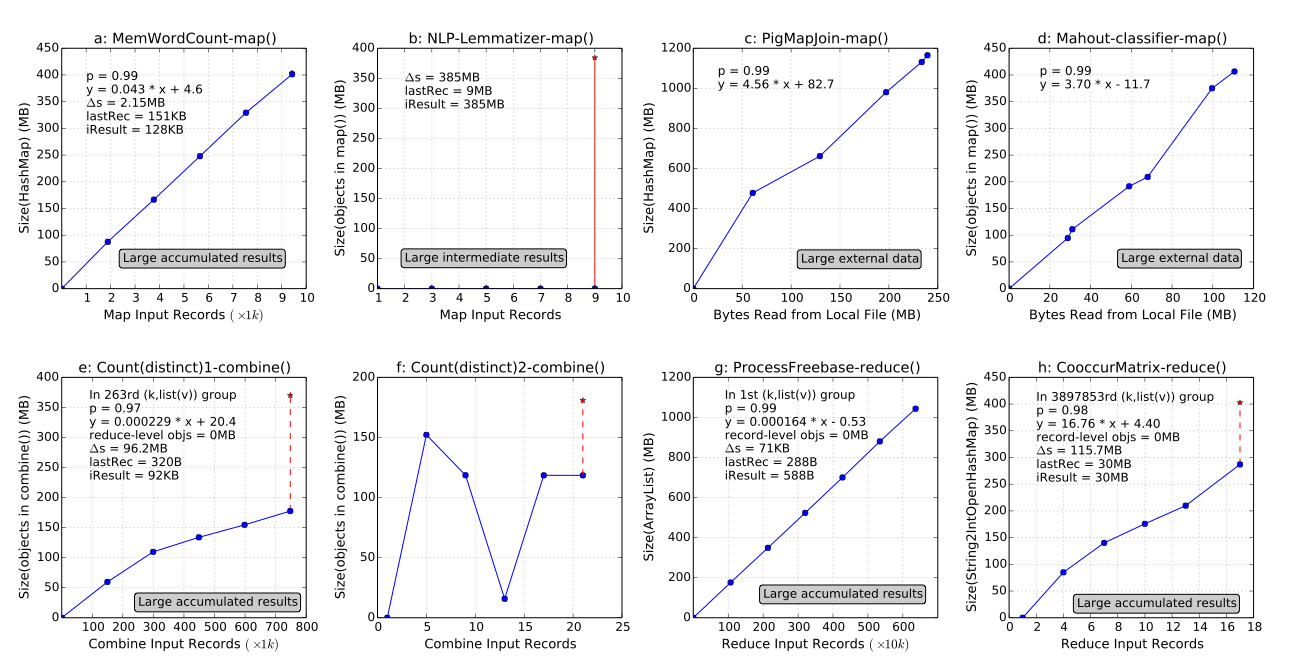
|  |  |  |
| --- | --- | --- |
| Rules | Cause pattern | Cause-related configurations |
| Memory growth in the last group is an outlier && record number in the last group is an outlier | Hotspot key | None |
| Large reduce-level accumulated results OR all the intermediate data that are shuffled from mappers are cached in memory | Improper data partition | reduce number and partition function |
| Fixed buffer exists && user code (i.e., map()) is nearly finished (e.g., has processed 80% of the input records) | Large fixed buffer | buffer size |
| Virtual buffer is filled with intermediate data OR user code (i.e., combine()) is nearly finished (e.g., has processed 80% of the input records) | Large virtual buffer | buffer size |

## 3. Evaluation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Job Name | Symptoms | Identified Cause Patterns | Cause-related Configurations | Verification and Real causes\* |
| Map phase:  Grep keywords | 1. No user objects  2. Large fixed buffer to be allocated | Large fixed buffer | buffer size | √ The fixed buffer is too large to be allocated in the heap |
| Map phase:  MemWordCount | 1. Linear trend in [K1, Kn)  2. map() has processed 30.3% records  3. a fixed buffer | Large map-level accumulated results | input data size | √ map() puts too many intermediate results (⟨word, count⟩ pairs) into a HashMap for aggregation |
| Map phase:  NLPLemmatizer | Sharp growth in Kn | Large intermediate results  Hotspot key | None | √ The input record (sentence) is large while the NLP API used in map() generates large intermediate results |
| Map phase:  Pig MapJoin | 1. Input records = 0  2. Large external data has been read  3. Linear trend between user objects and the external data | Large external data | None | √ map() reads large external data (a table) into a HashMap for fast JOIN |
| Map phase:  Mahout classifier | 1. Input records = 0  2. Large external data are read  3. Linear trend between user objects and the external data  4. a fixed buffer | Large external data | None | √ map() reads large external data (training data) into memory |
| Map&spill phase:  Count(distinct)1 | 1. Linear trend in Gn  2. Group Gn is an outlier  3. combine() has processed 99% records in Gn  4. a fixed buffer | 1. Large group-level accumulated results  2. Hotspot key | buffer size | √ Combine() puts the input records into a data structure called InternalDistinctBag to sort and deduplicate them. |
| Shuffle phase:  PigShuffle | 1. No user objects  2. All the intermediate data (map outputs) are cached in memory | 1. Large virtual buffer  2. Improper data partition | buffer size  partition number  partition function | √ All the map outputs are cached in the large buffer |
| Shuffle phase:  Count(distinct)2 | 1. Non-linear trend in Gn  2. combine() has processed 99% records in Gn  3. Large but not all the intermediate data are cached in the buffer | Large virtual buffer | buffer size | √、 Similar with Count(distinct)1 and many map output are cached in memory |
| Reduce phase:  ProcessFreebase | Linear trend in Gn | Large group-level accumulated results | None | √ reduce() puts lots of input records (text) into ArrayList for further use |
| Reduce phase:  CooccurMatrix | Linear trend in Gn  Group Gn is an outlier | 1. Large group-level accumulated results  2. Hotspot key | None | √ reduce() puts all the input records in a large group into a HashMap-like data structure for aggregation |

\*Real causes are identified by experts or based on our manual analysis of the source code.

Trend in the user code:

The blue line reveals the trend of “Size(user objects) vs. Input records” or “Size(user objects) vs. Size(external data)”. The solid red line reveals a real growth, while the dotted red line reveals the data structure related growth. Data structure such as ArrayList and HashMap can expand 1.5 or 2 times of the original size when they are nearly full. As a result, the data structure related growth is actually caused by large accumulated results. We rerun the use code and let is only process Kn/Gn to identify whether the growth in Kn/Gn is a real growth.