On the Impact of Memory Allocation on High-Performance Query Processing

Dominik Durner  
Technische Universität München  
dominik.durner@tum.de

Viktor Leis  
Friedrich-Schiller-Universität Jena  
viktor.leis@uni-jena.de

Thomas Neumann  
Technische Universität München  
thomas.neumann@tum.de

ABSTRACT

Somewhat surprisingly, the behavior of analytical query engines is crucially affected by the dynamic memory allocator used. Memory allocators highly influence performance, scalability, memory efficiency and memory fairness to other processes. In this work, we provide the first comprehensive experimental analysis on the impact of memory allocation for high-performance query engines. We test five state-of-the-art dynamic memory allocators and discuss their strengths and weaknesses within our DBMS. The right allocator can increase the performance of TPC-DS (SF 100) by 2.7x on a 4-socket Intel Xeon server.

ACM Reference Format:

1 INTRODUCTION

Modern high-performance query engines are orders of magnitude faster than traditional database systems. As a result, components that hitherto were not crucial for performance may become a performance bottleneck. One such component is memory allocation. Most modern query engines are highly parallel and heavily rely on temporary hash-tables for query processing which can be implemented in many different ways [2, 10, 15, 16]. As a result, a large number of short living memory allocations of varying size are requested. Memory allocators therefore need to be scalable and be able to handle myriads of small and medium sized allocations as well as several huge allocations simultaneously. As we show in this paper, memory allocation has become a large factor in overall query processing performance. New hardware trends exacerbate the allocation issues. Because most multi-node machines rely on a non-uniform memory access (NUMA) model, requesting memory from a remote node is particularly expensive.

In this paper, we perform the first comprehensive study of memory allocation in modern database systems. Although memory allocation is on the critical path of query processing, no empirical study on different dynamic memory allocators for in-memory database systems has been conducted [1]. Some online transaction processing (OLTP) systems try to reduce the allocation overhead by managing their allocated memory in chunks to increase performance for small transactional queries [4, 13, 14]. However, most database systems process both transactional and analytical queries. Therefore, the wide variety of memory allocation patterns for analytical queries needs to be considered as well.

Figure 1 shows the effects of different allocation strategies on TPC-DS with scale factor 100. We measure memory consumption and execution time with our multi-threaded database system on a 4-socket Intel Xeon server. In this experiment, our DBMS executes the query set sequentially using all available cores. Even this relatively simple workload already results in significant performance and memory usage differences. Our database linked with jemalloc can reduce the execution time to 2 in comparison to linking it with the standard malloc of glibc 2.23. On the other hand, jemalloc has the highest memory consumption and does not release memory directly after execution of the query. Consequently, the allocation strategy is crucial to the performance and memory consumption behavior of in-memory database systems.

2 EXPERIMENTAL ANALYSIS

The full version of the experimental analysis can be found in [3].

For the experimental evaluation, we use a database system that uses pre-aggregation hash tables to perform multi-threaded group bys and joins [10]. Our DBMS has a custom transaction-local chunk allocator to speed up small allocations of less than 32KB. Since only small allocations are stored within medium-sized chunks, the memory efficiency footprint of these small object chunks is marginal. Additionally, the memory needed for tuple materialization is acquired in chunks that grow as more tuples are materialized. Thus, we already reduce the stress on the allocator.

To simulate a realistic workload, we use an exponentially distributed workload to determine query arrival times. We sample
reduce the median latency to a similar level as TBBmalloc. However, the query latencies within the third quantile vary vastly. TCMalloc and malloc 2.23 are still not able to process the queries without introducing long waiting periods.

In Figure 4, we show the latencies for the $\lambda = 6$ q/s workload on different hardware architectures. On the single-socket Skylake X, all the allocators have very similar performance. Besides having more cores, AMD’s Threadripper uses two memory regions which requires a more advanced placement strategy to obtain fast accesses. In particular, TCMalloc and malloc 2.23 without a thread-local cache have a reduced performance. Yet, the most interesting behavior is introduced by the multi-socket Intel Xeon. jemalloc and TBBmalloc execute the queries with the overall lowest latencies and smallest variance. On the other hand, TCMalloc is worse by more than 10x in comparison to any other allocator. Both glibc implementations have a similar median performance but incur high variance such that a reliable query time prediction is impossible.

3 CONCLUSIONS

In this work, we provided a thorough experimental analysis and discussion on the impact of dynamic memory allocators for high-performance query processing. We highlighted the strength and weaknesses of the different state-of-the-art allocators according to scalability, performance, memory efficiency, and fairness to other processes. For our allocation pattern, which is probably not unlike to that of most high-performance query engines, we can summarize our findings as follows:

<table>
<thead>
<tr>
<th></th>
<th>scalable</th>
<th>fast</th>
<th>mem. fair</th>
<th>mem. efficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCMalloc</td>
<td>+/-</td>
<td>~</td>
<td>++</td>
<td>-</td>
</tr>
<tr>
<td>malloc 2.23</td>
<td>-</td>
<td>~</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>malloc 2.28</td>
<td>~</td>
<td>+</td>
<td>-</td>
<td>~</td>
</tr>
<tr>
<td>TBBmalloc</td>
<td>+</td>
<td>~</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>jemalloc</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

As a result of this work, we use jemalloc as the standard allocator for our DBMS.
REFERENCES


This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 725286).