Alexander Krause, Thomas Kissinger, Dirk Habich, Wolfgang Lehner

NeMeSys - A Showcase of Data Oriented Near Memory Graph Processing

Erstveröffentlichung in / First published in:


DOI: https://doi.org/10.1145/3299869.3320226

Diese Version ist verfügbar / This version is available on:

https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-806366
NeMeSys - A Showcase of Data Oriented Near Memory Graph Processing

Alexander Krause, Thomas Kissinger, Dirk Habich, Wolfgang Lehner
[firstname.lastname@tu-dresden.de]
Technische Universität Dresden
Dresden, Saxony, Germany

ABSTRACT
NeMeSys is a NUMA-aware graph pattern processing engine, which uses the Near Memory Processing paradigm to allow for high scalability. With modern server systems incorporating an increasing amount of main memory, we can store graphs and compute analytical graph algorithms like graph pattern matching completely in-memory. Our system blends state-of-the-art approaches from the transactional database world together with graph processing principles. We demonstrate, that graph pattern processing – standalone and workloads – can be controlled by leveraging different partitioning strategies, applying Bloom filter-based messaging optimization and, given performance constraints, can save energy by applying frequency scaling of CPU cores.

CCS CONCEPTS
• Information systems → Graph-based database models; Main memory engines; Data scans; Point lookups; Relational parallel and distributed DBMSs.

KEYWORDS
Graph, In-memory, Bloom filter, NUMA

1 INTRODUCTION
Processing the ever-growing data volume demands for continuously increasing server capacities. These are usually achieved by implementing several multiprocessor inside one scale-up system which exhibit a non-uniform memory access (NUMA), where the processors memory domains are still accessible through an interconnect network. Modern relational database systems are capable to exploit such server systems by employing the state-of-the-art data-oriented architecture (DORA), which has been proven to show superior scalability. Another key finding of recent research is the application of online energy controlling mechanisms as described in [5]. Following these findings, relational in-memory database can achieve energy savings under given performance constraints by scaling the core- and uncore-frequencies or turning off cores completely.

A still emerging use case for in-memory databases is the processing of graphs. Aside many other algorithms, graph pattern matching is an important application as described in [6]. Graph pattern matching (GPM) is a declarative, topology-based querying mechanism [4] with considerable complexity. Queries are usually entered as a graph itself and the systems task is to find all combinations of vertices, which match to the constraints of the query vertices. Computing GPM on NUMA systems is challenging, due to its inherent random memory accesses and thus an efficient processing management is needed to maintain performance.

To overcome these challenges, we present NeMeSys, a DORA based graph processing engine, which leverages well known principles from the relational to efficiently process GPM queries on NUMA systems. In addition to the DORA, NeMeSys features (I) graph specific partitioning methods, (II) specifically tailored graph operators such as Vertex-Bound or Edge-Bound and (III) a fine tunable messaging optimization.

In our demonstration, we want to showcase how different configurations take effect on the systems performance. We will not only show isolated effects but the combined influence of the set of graph specific system characteristics. This way we aim to underline the importance of careful system tuning, which is yet still prone to workload changes. Additionally, we can show that our previously developed energy adaptivity
mechanisms for \textit{relational} in-memory databases from [5] are
general enough to be seamlessly integrated into NeMeSys,
which specifically targets graph processing.

2 \textbf{SYSTEM}
From a hardware perspective, the scale-up approach is mainly
characterized by the separate memory domains and the result-
ing NUMA behavior. NeMeSys is built upon the well-
known DORA and leverages near-memory computing (NMC)
principles with its basic architecture being portrayed in Fig-
ure 1. NMC means, that we limit the scope of each worker
to memory domains, which are directly connected to their
socket (local instead of remote accesses).

2.1 NeMeSys Storage Layer
Generally, we store a \textit{directed edge-labeled multigraph} in
NeMeSys as a set of triples containing only the source ver-
tex, the target vertex and the label of the edge between these
two vertices\footnote{Property graphs can also be expressed as directed edge-labeled multigraphs by introducing additional vertices and edges for every property and its value.}. This layout can be efficiently stored in a rela-
tional table with exactly these three columns. Considering
only \textit{outgoing edges} is quite sufficient to fully represent the
 topology of the graph on the storage layer, thus leading to
the naming of our data container: the \textit{outgoing edge table}.

The core concept of DORA is that all data objects are im-
plcitly partitioned and partitions are exclusively accessed
by the assigned worker thread. That means, we have to parti-
tion our \textit{outgoing edge table} and we decided to partition this
 table based on the source vertices, wherefore we apply our
guidelines from [6]. On the one hand, the advantage is that a
worker thread is able to exclusively process several vertices
with all of its outgoing edges within a partition without us-
ing latches or locks for synchronization. On the other hand,
da disadvantage is that we implicitly replicate vertices, but
the overhead is negligible due to dictionary encoded vertices.
Of course, in addition to our \textit{outgoing edge table}, we also
need to store the partitioning information as presented in
Section 2.3.

2.2 NeMeSys Processing Layer
\textit{Conjunctive queries (CQ)} are a well-studied mechanism for
expressing graph patterns [7] and NeMeSys uses this mech-
anism. A CQ decomposes the pattern into a set of \textit{edge pre-
dicates} each consisting of a pair of vertices and an edge label.
An easy example for a CQ would be a query containing three
vertices and two edges: (A) \textit{knows} (B) \textit{livesIn} (C). This
query wants to find all combinations of vertices, where A
\textit{knows} B and B \textit{livesIn} a place C.

NeMeSys accepts a query string of multiple triples as
CQ, where each triple represents an edge predicate. Each
 triple contains a comma delimited list of source vertex, target
vertex and an edge label and all three positions can be either a
variable as placeholder or a previously known \textit{id}, if a specific
vertex or label shall be part of the query pattern.

We identified three graph specific operators, which are
necessary to execute conjunctive queries:
\textbf{Scan Operator.} The Scan operator performs a parallel
vertex scan over all partitions in the case that the starting
as well as the target vertex of a CQ triple are unknown.
By specifying a certain edge label predicate, the operator
returns only edges with the specific label. The Scan operator is always the first operator in the pattern matching process.

**Vertex-Bound (VB) Operator.** The VB operator takes an intermediate GPM result as input and tries to match new vertices in the query pattern according to the following CQ triple. The operator has to be only applied when either the source vertex or target vertex is known and thus bound.

**Edge-Bound (EB) Operator.** This operator ensures the existence of additional edge predicates between known vertex matching candidates of the CQ. It performs a data lookup with a given source and target vertex as well as a given edge label. If the lookup fails, both vertices are eliminated from the matching candidates. Otherwise the matching state is passed to the next operator or is returned as final result. During query processing, workers will continuously process all of their partitions incoming messages with the applicable operator. Moreover, each operator in NeMeSys is asynchronously processed in parallel and generates new intermediate results that invoke the next operator in the pattern matching query. Hence, NeMeSys allows for intra- and inter-query parallelism as well as pipelining to fully exploit the high number of cores in scale-up systems.

### 2.3 NeMeSys Infrastructure Layer

To efficiently support the above described pattern matching processing, NeMeSys provides two important components at the infrastructure layer:

**Partition Manager:** The partition manager keeps track of the edge-to-partition assignments and exploits, that all outgoing edges of one vertex are always stored in the same partition. Basically, the partition manager is maintaining a one dimensional index, which contains all vertex ids as keys and the target partitions as values. This allows NeMeSys to easily identify the correct partitions, on which a worker needs to apply the next query operator. Because of very frequent index requests, its size should fit in the CPU cache.

**Message Interface:** Because of the enforced locality and the asynchronous execution, workers need to communicate the query pattern matching state (intermediate results) via messages instead of directly enqueuing it at target partitions. In NeMeSys, every partition has an incoming and outgoing message queue. A coordinator is continuously performing cross socket copy operations, to move the messages of one worker to all corresponding target partitions. The messaging itself can also become a performance bottleneck, because not all CQs contain only forward oriented edges. Since we only index the source vertex of an edge, targets of backward oriented edge predicate requests can not be answered through an index lookup. In such cases, we need to send a broadcast targeting all data partitions to scan through their target column and search for the requested vertex.

### Bloom filter optimization.

Since NeMeSys is an asynchronous system, communication is performed via messages through the aforementioned Messaging Interface. Generally speaking, avoiding communication, such as the previously mentioned broadcasts, as much as possible can only increase the performance. Hence, we employ a Bloom filter-based message optimization, which serves as a secondary index to prevent unnecessary messages, like vertex- or edge-requests targeting partitions, which do not actually contain them. Since we use dictionary encoded integer keys for our partitioning, we can employ a fast and simple hash function inside the Bloom filter. Usually, a modulo operator is used to hash integer ids to a bucket. However, the modulo operator is very costly [1]. Thus we exploit the residue class ring property, where:

\[(x)_{base} \pmod{base^k} = \text{last k digits of } (x)_{base} \cdot \phi(base^k)\]  

For 64 bit unsigned integer vertex ids, which are of base 2, the last k digits of \((x)_2\) are given by applying a bitmask, or bitwise and, of \((2^k − 1)\) to \(x\).

Following this approach, we can replace the modulo operator by bitwise and while selecting prime numbers for \(a_i\), since they yield best uniformity results [2], and use \(2\) as our Bloom filter hash function.

\[H_i(x) = y = (a_i \cdot x) \pmod{(M − 1)}\]  

### 2.4 Energy adaptivity

As stated in Section 1, we integrate the energy management from [5]. For this purpose, NeMeSys implements software sensors, which measure the time spent performing different tasks, e.g. actual computing time or query latency and the energy consumption is measured through the processor integrated RAPL counters [3]. We are able to specify a performance constraint, i.e. a maximum query latency and allow NeMeSys to set the frequencies of all cores accordingly. Turning off cores, when there is just not enough computational demand, is allowed by the loosely coupled DORA approach, which was introduced in [5]. In a strict DORA, workers are only allowed to touch their very own data partition, but we allow them to process any data partition, which is currently not being processed.

### 3 DEMONSTRATION SETUP

With our demo, we will showcase our DORA based graph processing system NeMeSys. The demo itself is split into two parts. First, we want to show the influence of graph partitioning and the resulting data allocation on the intra-query parallelism of graph pattern matching inside NeMeSys. We will explain the query generation and justify the graph specific operators by showcasing applicable scenarios. Following the basics, we will highlight the importance of the
messaging interface and the performance impact of unnecessary messages. Demo visitors can play with all knobs and observe the corresponding system behavior (cf. Fig 2).

Adjusting the control knobs. The performance of graph processing is influenced by the underlying partitioning, as shown in [6]. Thus, we allow the user to apply different graph partitionings like e.g. Hash or Multilevel k-Way. In addition, we provide multiple datasets based on bibliographic, social or protein network characteristics. We will present allocation mappings, which underline and explain the differences between certain partitioning strategies. For optimizing the messaging, we consider Bloom filters. Their false positive rate is tightly bound to the size of the internal bitfield. Users can change the number of available bits to trade off memory consumption for messaging efficiency.

Controlling Energy. Within this second part of our demo, we allow the user to play around with the aforementioned control knobs and view their influence on the energy consumption of the system. This includes starting different workload scenarios for a given configuration while allowing for online adaption for energy efficiency, as introduced in [5].

After selecting the desired configuration including the workload pattern, the benchmark can be executed and its results will be presented below. We visualize inter-socket communication as well as computational demand of the sockets themselves for better comparability. Furthermore, key characteristics like energy demand or average query latency are displayed. This allows the user to run a plethora of different benchmark workloads and compare the impact of the applied optimization strategies.

4 CONCLUSION AND FUTURE WORK

With our demo, we present NeMeSys, which allows for NUMA-aware, energy-adaptive graph processing on modern multiprocessor systems. Furthermore, we underline the importance of a careful yet workload dependent parameter selection. Our vision is to build an adaptive system, which can decide the optimal data partitioning as well as Bloom filter sizes for a given graph and workload and adapt to it on the fly. However, graph partitioning is a hard problem. Thus, we need to identify suitable heuristics and transformation methods, to change one partitioning layout to another, which requires intensive future research.

5 ACKNOWLEDGEMENTS

This work is partly funded within the CRC 912 (HAEC).

REFERENCES