HUGS – A Lightweight Graph Partitioning Approach

Alexander Krause, Hannes Voigt, Wolfgang Lehner
Technische Universität Dresden
Database Systems Group
Dresden, Germany
{firstname.lastname}@tu-dresden.de

ABSTRACT

The growing interest in graph data lead to increasingly more research in the field of graph data management and graph analytics. Nowadays, even large graphs of upto a size of billions of vertices and edges fit into main memory of big modern multisocket machines, making them a first-grade platform for graph management and graph analytics. High performance data management solutions have to be aware of the NUMA properties of such big machines. A data-oriented architecture (DORA) is a particular solution to that. However, it requires partitioning the data in a way such that inter-partition communication can be avoided.

Graph partitioning is a long studied problem and state-of-the-art solutions, such as multilevel k-way partitioning and recursive bisection achieve good results in feasible time. Integrating such solution is a rather difficult task, though. In this paper, we present a more lightweight approach called HUGS. The key idea of HUGS is to reuse the BFS routine present in a graph data management system anyway, since it is the foundation of many analytical graph algorithms. HUGS is not meant to produce a good general-purpose graph partitioning but good runtimes of BFS graph traversals such as reachability queries on DORA systems. In our experiments HUGS showed capable of finding good graph partitionings faster than state-of-the-art approaches. The partitioning found by HUGS also showed shorter runtimes for reachability queries.

Keywords

Graph, Graph Partitioning, Reachability, BFS, DORA

1. INTRODUCTION

The last decade has seen resurgence of interest in graph data management [2]. With the network data model in the 1970s [28] and object-oriented database systems in the early 1990s graph-based data models and graph query languages got considerable attention in research already [3]. However, the traction of today's graph data management efforts is unequally higher with many major IT companies and DBMS vendors on the band wagon [7, 24]. Among others, one major driver behind the graph concept's revival is a shift in the interest of analytics from merely reporting towards data-intensive science and discovery [11]. Graph data can easily range in the size of billions of vertices and edges. Prominent examples of large graphs are the Facebook friendship graph, the Twitter follower graph, citation networks, web link networks, road networks, supply chains, etc. (cf. [21]). The analysis even of the biggest graphs is often done with recursive algorithms [26]. The Breadth First Search (BFS) is one of the most fundamental building blocks for many popular graph analysis algorithms, such as algorithms to determine reachability, connected components, and betweenness centrality [4, 13, 19].

The approach on how to work with a given graph is always determined by its size. The storage and mining of such potentially huge data sets requires computational resources ranging from small workstations up to whole compute clusters. With today's continuously decreasing main memory cost and increasingly stronger hardware, even standalone server workstations with multiple CPUs and terabyte of main memory can process graphs of the billion scale. Such multiprocessor machines consist of multiple sockets, whereby every socket holds multiple CPUs and a share of the whole systems main memory. In these large shared memory machines special attention has to be payed to the organization of memory allocation and access [1, 10, 18]. Each socket manages its share of physical memory and provides the other sockets access. Although each core has access to each byte of memory, it is a Non Uniform Memory Access (NUMA). Bandwidth and latency is lower for remote memory access to other sockets than the access to a socket's own memory. Data management systems oblivious of these NUMA effects take huge performance penalties on a large scale for remote memory accesses [18]. The Data-Oriented Architecture (DORA) [23] is an architecture for so called NUMA-aware DBMS. Such DBMS try to co-locate data and process as much as possible locally to avoid expensive remote memory access [20]. In particular, partitions of the data, are bound to an individual socket, and processed only by threads running on this socket exclusively. Thereby remote memory access is limited to inter-partition communication of operators such as a traversal over a graph. Obviously, the partitioning has a huge impact on how much inter-partition communication takes place during processing of a specific operator. In terms of graph processing on multiprocessor systems, the partitioning of the graph is inevitable.

28^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 24.05.2016 - 27.05.2016, Nörten-Hardenberg, Germany.

Partitioning graphs is a well studied field [5]. The typical problem definition asks for the partitioning of a graph into k partitions so that the number of edges cut at partition borders is minimal. In this absolute form the problem is NP-hard. Over the years many heuristic algorithms have been developed that try to minimize edge cuts. Prominent examples are recursive bisection [14] and multilevel k-way partitioning [16]. By exploiting the graphs topology, these general purpose algorithms find good and useful partitionings in many real world scenarios in feasible time. However, the resulting partitions are not optimized for BFS-based analyses on a multiprocessor system.

In this paper we propose a novel lightweight graph partitioning approach called HUGS. HUGS determines graph partitioning with the help of BFS traversals starting from high-degree vertices. As we reuse the BFS traversal operations, which are available in graph data management system anyway, HUGS only needs minor additions to work with common graph systems. In contrast to the general purpose partitions of current graph partitioning approaches, HUGS exclusively aims at a partitioning scheme good for the speed-up of BFS-based reachability queries in a NUMA setting. HUGS offers two advantages. First, it provides a suitable partitioning for BFS-based analyses and second, its lightweight design allows an easy integration into standard graph systems. In our experimental evaluations HUGS exhibits partitioning speed and partitioning quality for reachability queries comparable to the state-of-the-art algorithms. Hence, HUGS offers a significantly easier to implement, lightweight alternative to state-of-the-art graph partitioning algorithms for graph data management system.

In the remainder of the paper we first describe the DORA-based database infrastructure which we are targeting at (Section 2) and detail on the graph partitioning problem as well as state-of-the-art graph partitioning algorithms (Section 3). We continue with presenting our lightweight partitioning approach HUGS (Section 4) and the experimental evaluation (Section 5). Finally, we discuss related work (Section 6) and conclude the paper (Section 7).

2. PROBLEM STATEMENT

As described in the previous section, we want to optimize BFS-based graph analytics for NUMA affected multiprocessor systems. Since DORA is well tailored for the NUMA effect, it is predestined for exploiting locality in graph analysis on a NUMA system.

For most systems, execution threads or transactions are the central point of view, one does also say thread-to-transaction assignment. Transactions need to be globally synchronized and locks and latches need to be implemented, in order to prevent the system from anomalies, such as concurrent writings to the same data record. In a DORA system, where the thread-to-data assignment holds, no specific locks are needed [23]. Inside a database system based on DORA, each processing unit of the NUMA system represents a worker [20]. During computations, the worker should never switch its affinity to another physical processing unit on another socket, since this would introduce additional messaging overhead. Every worker is assigned with one or multiple data partitions of the whole data set. This approach holds, regardless if the system stores graph or relational data. One advantage of this architecture lies in the implied parallelism of the system. When a transaction needs to access data from multiple partitions, e.g. for a join, every worker can independently prepare sub-results for its available data partitions. Once these local computations have been concluded, requests to other units can be made for fetching possible join partners. As every worker is solely working on its own partitions, no further protection for the data partitions needs to be considered.

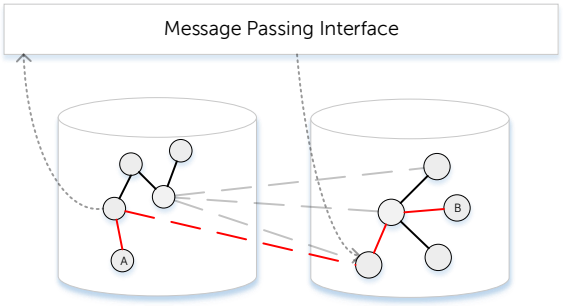


Figure 1: Communication between workers

Cross-partition operations require communication between the actual workers. This effect occurs, if e.g. a traversal operation accesses a part of the graph, which is stored in another partition than it is currently running in. For this purpose, the system needs to provide an asynchronous, high-throughput message passing layer that allows to hand over a task to another worker. For instance, when a traversal query reaches a border vertex inside a partition of one worker and the neighbor of this vertex resides in one partition of another worker, a message with the current search status and the target vertex will be sent to the second worker, asking to resume the search from this point on. This process is illustrated in Figure 1.

As the message passing layer is aware of the partition assignments, it will directly send messages to the required workers [20]. Every message constitutes a remote memory access, which extends to any NUMA system. Assuming a balanced utilization of all sockets, a low number of messages typically reduces the total runtime of a given workload. Obviously, the partitioning of the graph has significant influence on how many message will have to be sent for a given traversal.

When performing a BFS, the search starts from one specific vertex. Then, all its subsequent neighbors will be visited following the edge direction. Undeniable, the BFS will eventually reach a partitions border, which leads to communication overhead. There are two scenarios to consider. (1) Staying as long in the current partition as possible to avoid any communication at all and (2) reaching partition boundaries as fast as possible in order to exploit the system inherent parallelism to the maximum. The effectiveness of either strategy is dependent on the current system load. If other workers are highly utilized, completely searching through the current partition before notifying others may produce a better runtime behavior than sending messages to other workers as soon as a border vertex has been found. However, if the system is in an idle state, the total opposite is needed. Residing in the current partition as long as possible would completely underutilize the system ressources. Therefore,

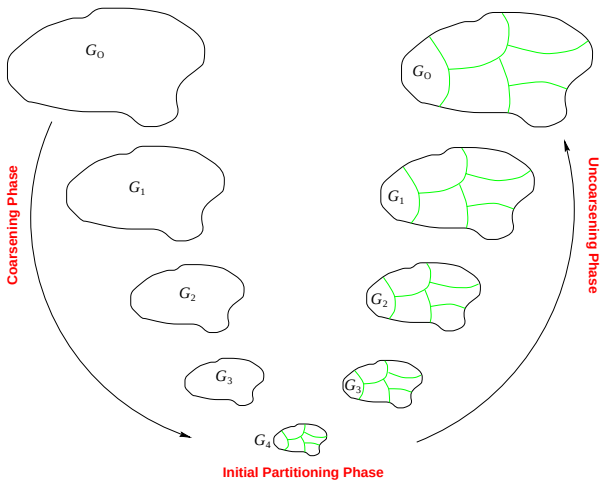


Figure 2: Multilevel k-Way partitioning scheme from [16]

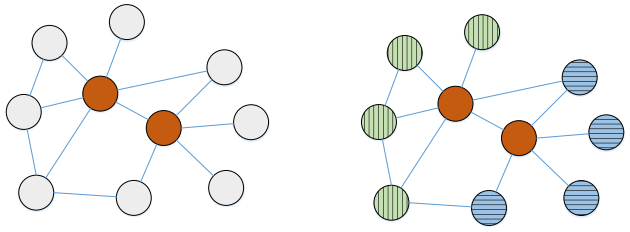
sending as much messages as possible to other workers can only increase the query performance.

3. CLASSICAL GRAPH PARTITIONING

In this work we are considering a graph $G = (V, E)$, where $v \in V$ is a vertex in G and $(v_i, v_j) \in E$ with $E \subseteq (V \times V)$ is an edge from v_i to v_j. We are only considering lossless partitionings into disjoint partitions $P_1, P_2, ..., P_n \subseteq V$ with $P_i \cap P_j = \emptyset$ for all $i \neq j$ and $i, j \in [1, n]$ as well as $\bigcup_i P_i = V$.

The Graph Partitioning Problem (GPP) is well studied. Dividing a graph G into k partitions of balanced size, is known to be NP-complete [12]. Practically feasible solutions can only solve the problem heuristically. Many approaches have been developed for tackling the GPP. One of them is the yet most successful multilevel partitioning [5]. Multilevel partitioning itself is no actual algorithm, rather a heuristical strategy. It consists of three stages in which multiple methods can be applied. The three stages are (1) coarsening the input graph, (2) finding a partitioning for the coarsened graph and (3) uncoarsening the partitioning back to the granularity of the input graph (cf. Figure 2). Basically, the idea is to do the actual partitioning only on graph sizes where complexity of the partitioning problem is bearable. The coarsening is done by combining multiple vertices into one abstract vertex. Edges are coarsened likewise by maintaining the graphs topology as well as possible. Typically, the coarsening is repeated until the graph is small enough (a few hundreds of vertices). On this very small graph even expensive partitioning algorithms can be applied without much runtime burden. Finally, the uncoarsening iteratively unpacks the abstract vertex. After each uncoarsening a refinement procedure optimizes the partitioning of the now finer grained graph. Since the refinement steps start from an already well partitioned base, the added overhead remains small.

There are two approaches for multilevel partitioning, namely k-way [16] and recursive bisection [14]. The main difference for both methods resides in the second phase. Multilevel k-way partitioning aims to partition the graph into k partitions directly. While recursive bisection recursively divides the graph and the resulting subgraphs into two partitions, until the final number of partitions is reached. Both require graphs to be undirected, since these approaches aim to reduce the edges cut between partitions.



(a) Top 2 Hubs, 1 per partition

(b) Final Result

Figure 3: Partitions resulting from HUGS

For BFS-based analytics, these partitionings will not be optimal. As undirected graphs can always be traversed in any direction along every edge, none of the vertices can become a dead end, as long as it is part of more than one edge. For directed graphs however, any vertex could represent a sink, i.e. when there are only ingoing edges from other vertices. This key characteristic has to be considered when partitioning directed graphs, especially for BFS-based queries.

4. HUGS – HUB-CENTERED GRAPH PARTITIONING

We claim, that partitioning a graph with a BFS-based approach will increase the performance of BFS-based queries. The idea behind our novel graph partitioning method is, that many paths between two vertices will make use of high degree vertices. Additionally, geometrical patterns like triangles or chains are likely to pass through these nodes. Therefore, centering the partitioning around *hubs* of the graph should create partitions, which provide sufficient locality for certain queries. The resulting partition will most likely represent the structure which would be created when performing a BFS on the graph. Having the partition equally structured as the search tree itself should lead to a significant synergy between both.

HUGS is a lightweight BFS-based graph partitioning approach. As many graph systems almost always implement a version of BFS, this implementation can be reused with slight modifications. In contrast to the multilevel partitioning methods, HUGS considers the whole graph without coarsening it. Its heuristic is based on centering the partitions around so called hubs in the graph. A hub is a vertex with a high degree, i.e. it has many incoming and outgoing edges. The intuition behind this heuristic is that hubs, because of the many incoming and outgoing edges, are most likely part of many paths in the graph – and particularily of paths in the neighborhood of hubs. Essentially, HUGS runs BFSs starting from these hubs to determine the partitions. The order in which the vertices are relaxed during the BFS determines the partitioning time as well as the partitioning quality.

Algorithm 1 shows HUGS' partitioning procedure. A set of hubs H is maintainted besides the base data. H descendingly lists the vertices h with a top-$(k \cdot n)$ degree $deg(h)$, where $k \in \mathbb{N}^+ \wedge k \cdot n \ll |V|$. We define the degree $deg(v)$ of a vertex v as $deg(v) = |K(v)|$ where $K(v) = \{w \mid (v, w) \in E \vee (w, v) \in E\}$ is the neighborhood of v. Initially, HUGS adds k of the top-n hubs and their

Algorithm 1 HUGS

```
1: H ← A Set of Hubs
2: P_i ← A Partition
3: N ← A list of partition candidates
4: for all P_i do
5:    Add k of hubs in H as root hubs to P_i
6:    Add the neighbors of the root hubs to P_i
7:    while P_i not full do
8:       for all v ∈ P_i do
9:          for all u ∈ K(v) ∧ u not visited do
10:            Set u visited
11:            deg_{P_i}(u) = |K(u) ∩ P_i|
12:            N ← N ∪ {u}
13:         end for
14:      end for
15:      Order N by deg_{P_i}(u) / deg(u)
16:      Add top-t of N to P_i and remove from N
17:      Set N \ P_i unvisited
18:   end while
19: end for
20: Add unreachable nodes to the smallest partition
```

direct neighbors to each partition (line 5–6). We refer to these k top-n hubs as the root hubs of the partition. In each BFS step, HUGS does not add all newly explored vertices to the current partition. Instead, it determines for each newly explored vertex u the ratio of its neighbors in the current partition $deg_{P_i}(u)$ and all its neighbors $deg(u)$ (line 8–15). The ratio is a heuristical local measure how close the vertex is to the current partition. Newly explored vertices with a top-t ratio are added to the partition (line 16).

There are two parameters to steer the performance of HUGS: the number of root hubs k and the maximum growth rate t. Starting with only one root hub creates a partition which is coherent but HUGS needs more BFS steps to fill the partition. Multiple root hubs result in a partition consisting of a scattered number of sub areas in the graph which usually results in faster partitioning time but yields less quality. A higher growth rate means that more of the explorer vertices are added to the partition, which drastically reduces the partitioning time but yields less quality, since many low ranked vertices are added to the partition as well.

Figure 3 illustrates the whole process for n = 2 partitions and serves as an example. First, the two vertices with the highest degree in the graph are selected, as shown in Figure 3(a). Starting from the vertex with the higher degree, all neighbors will be scanned and added to its partition. If the maximum number of vertices for this specific partition has been reached, the BFS will terminate. Otherwise the search continues with the neighborhood sets of the already selected vertices. The same procedure is analoguesly applied to the second hub.

HUGS provides two advantages. The first advantage applies to the optimized partitioning. Creating synergies between the data partitioning and the search routine increases query performance. Second, through reusing existing implementations of BFS, HUGS is lightweight and easy to implement. However, a drawback of our method can result from the graphs topology. If the highest degree vertices are all located in each others neighborhood set, the algorithm may produce imbalanced results.

5. EVALUATION

To show the benefit of our novel and lightweight graph partitioning approach, we conducted a series of experiments for testing a prototypical implementation of HUGS. As data we used two directed graphs taken from the online graph data collection of SNAP [21]. The first graph is the *Wikivote* graph. It represents the voting behaviour of users, who had to elect new moderators. It is a rather small graph with 7115 vertices and 103689 edges. The second graph is the *Stanford* web graph, which represents the structure of the Stanford University website with 281903 vertices and 2312497 edges. Every edge represents a hyperlink between two webpages.

As for our test setting, we partitioned every graph and measured different statistics. For the multilevel k-Way partitioning and recursive bisection algorithms, we used the METIS library [17]. Additionally we also used a random partitioning as a baseline. For every partitioning appraoch we measured the runtime of the partitioning algorithm as well as the runtime of the ten random reachability queries. All tests have been performed using a workstation with an AMD Opteron 6274 CPU and 64 GB of main memory. Because of its significantly longer query runtime, the performance of the random partitioning was considerably worse than all other approaches. Therefore we omited its runtimes from the corresponding figures. In order to maintain comparability and to avoid hidden latencies, we performed the testing sequentially and measured the query time spent per partition. Since HUGS outperforms its competitors in a sequential environment, it is obvious that a DORA system would also show better performance values for HUGS compared to multilevel k-Way and recursive bisection.

Figure 4 and 5 show the partitioning times of HUGS compared to multilevel k-Way and recursive bisection. For Wikivote, HUGS outperforms both approaches due to its reduced effort in creating topologically balanced partitions. For Stanford, HUGS struggles with a smaller number of partitions. A full BFS with continuous ranking of all neighbors of the partition leads to longer partitioning times. With a growing number of partitions, HUGS' partitioning time drops a bit while the k-Way partitioning and recursive bisection algorithms take considerably longer. For DORA systems, which aim at large multiple socket server machines, partitionings into eight and more partitions are the relevant scenarios. Here, HUGS consistently outperforms its competitors.

For query runtime tests, we selected ten pairs of random vertices in the graph. These pairs were fixed for all partitioning approaches as well as for every number of partitions. The reachability traversal is implemented as BFS as well. Starting from the source node of the reachability request, the search procedure starts forward directed. As explained in Section 2, our data is stored on one specific worker. Therefore, we try to find the target vertex in the root partition. When the BFS reaches an inter-partition edge, the worker with the partition containing the targeted vertex will be notified and immediately starts his own search procedure. When multiple vertices have an inter-partition edge to the same target partition, these targeted vertices will be queued with the corresponding worker. If those vertices are already visited from the current or already terminated search, no additional search will be scheduled. The procedure terminates, when one of the workers finds a path from a border vertex to the target vertex. This experiments represent the first of the two scenarios explained at the end of Section 2.

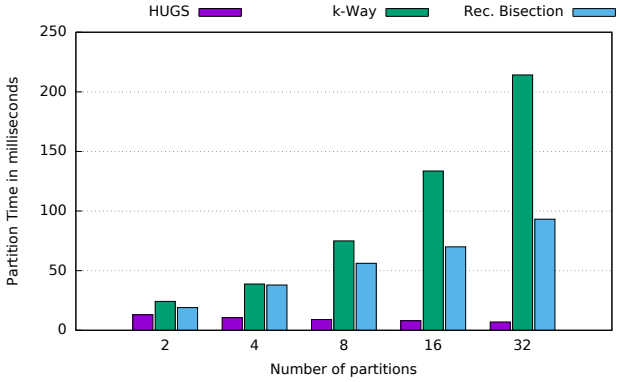


Figure 4: Partitioning times for the Wikivote Graph

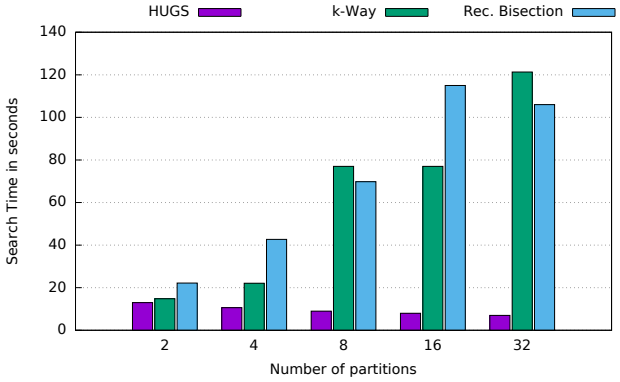


Figure 6: Reachability runtimes for the Wikivote Graph

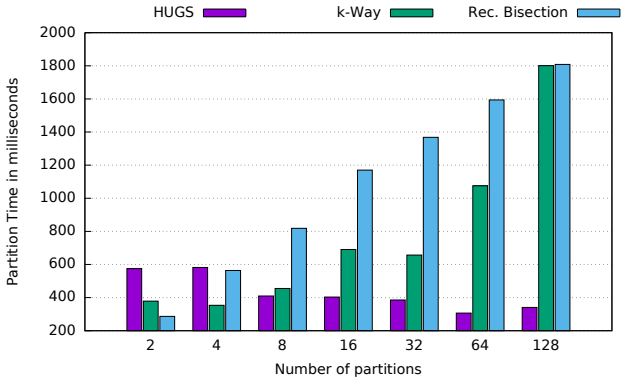


Figure 5: Partitioning times for the Stanford Graph

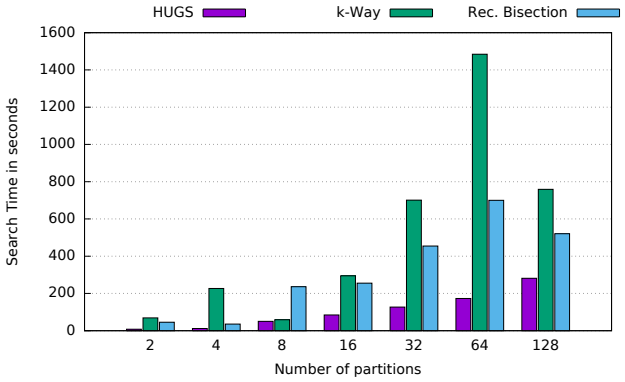


Figure 7: Reachability runtimes for the Stanford Graph

The runtimes of the reachability queries are displayed in Figures 6 and 7. The small Wikivote graph shows, that a search based on HUGS provides consistently faster answers. The topological partitionings of k-Way partitioning and recursive bisection tend to introduce more overhead for a BFS-based search. Since the partitioning and search scheme are both BFS-based, they can synergize and yield good performance values, compared to the two other approaches.

6. RELATED WORK

Many works aim at improving graph partitioning algorithms themselves. Karypis and Kumar [15] are improving the multilevel partitioning approach by parallelizing multiple steps. At first, the bisection of the subgraphs is performed by two processors. The resulting partitions are to be bisected as well. This workload is also split up for multiple processors. The authors refer to these techniques as parallelism in the recursive and in the bisection step. HUGS is designed as a single threaded BFS-like partitioning. Slota, Rajamanickam and Madduri show, that the BFS itself can be parallelized [27].

Ding et al. are discussing a min-max cut problem for data clustering [6]. The authors claim that they achieve a balanced partitioning with maximizing the similarity between vertices in the same partition. In contrast the similarity or association to vertices of other partitions is to be minimized. Other approaches like normalized cut or ratio cut are said to be less efficient. Generally speaking, this approach seeks to cluster identities, with high similarity and thus partitions the graph on a logical level. In contrast, HUGS partitions the graph on a topological level. We exploit the fact, that high degree vertices are most likely to be visited, when searching connections between two vertices.

A different field of applications is to model problems as graphs and apply partitioning algorithms on it to simplify computations on this data. Fern and Brodley use multilevel graph partitioning to solve the cluster ensemble problem [8]. They map the datapoints to vertices and partition them according to their similarity, which is used as an edge weight.

Gilbert and Zmijewski show, that using the Kernighan Lin Algorithm [22] can improve the performance of message passing multiprocessor systems, like the hypercube [9]. Similar to our work, the ultimate goal is to reduce communication overhead between the workers. As our method is based on BFS only, it will most likely not produce optimal partitionings with a minimal edge cut. Yet we produce highly localized partitions, which reduce the communication overhead of BFS-based reachability queries.

Schloegel, Karypis and Kumar [25] as well as [29] use graph partitioning for exploiting parallelism on meshes, which are often the basis of scientific calculations. Meshes can consist of massive amount of data and therefore don't fit into a single machines main memory. Since scientific calculations often refer to incremental updates of neighboring nodes, we assume that HUGS could be applied to this field as well.

7. CONCLUSIONS AND FUTURE WORK

We presented HUGS, a lightweight, BFS-based graph partitioning algorithm. HUGS is meant for partitioning graph data

for its management in DORA systems. We showed that HUGS yields better performance for BFS-based reachability queries than its state-of-the-art graph partitioning competitors. At the same time HUGS also determines the partitioning quicker than these standard approaches particularly for relevant scenarios of 8 and more partitions. Implementing HUGS in a graph data management system is simple since its can build on existing BFS code, which is one of the most fundamental graph routines and can be assumed to be implemented by every typical graph data management system in a highly optimized fashion.

Our investigations reported here shows HUGS to be a promising approach for DORA systems. As our prototype mostly favors the first partition, we are investigating other filling techniques, e.g. an alternating calculation of each partition, whereby the corresponding runtime penalties are yet to be resolved. We are further investigating other heuristics to determine for the partition growth phase. Since HUGS exploits two parameters for adjustments, it is customizable. However, the algorithms performance is dependent of the underlying graphs structure. Therefore, we need to find good heuristics to automatically determine upfront, which configuration works best for a given graph. Further down the road, we will look into an incremental version of HUGS for maintaining the partitioning upon updates to the graph.

8. ACKNOWLEDGMENTS

This work is partly funded by the German Research Foundation (DFG) within the Cluster of Excellence "Center for Advancing Electronics" (cfaed) in the Collaborative Research Center 912 *Highly Adaptive Energy-Efficient Computing* (HAEC).

9. REFERENCES

[1] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood. Dbmss on a modern processor: Where does time go? In *VLDB 1999*, pages 266–277, 1999.

[2] R. Angles. A comparison of current graph database models. In *ICDE 2012*, pages 171–177, 2012.

[3] R. Angles and C. Gutiérrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1), 2008.

[4] A. Z. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener. Graph structure in the web. *Computer Networks*, 33(1-6):309–320, 2000.

[5] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent advances in graph partitioning. *CoRR*, abs/1311.3144, 2013.

[6] C. H. Q. Ding, X. He, H. Zha, M. Gu, and H. D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *IEEE 2001*, pages 107–114, 2001.

[7] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner. SAP HANA database: data management for modern business applications. *SIGMOD Record*, 40(4):45–51, 2011.

[8] X. Z. Fern and C. E. Brodley. Solving cluster ensemble problems by bipartite graph partitioning. In *(ICML 2004)*, 2004.

[9] J. R. Gilbert and E. Zmijewski. A parallel graph partitioning algorithm for a message-passing multiprocessor. *International Journal of Parallel Programming*, 16(6):427–449, 1987.

[10] N. Hardavellas, I. Pandis, R. Johnson, N. Mancheril, A. Ailamaki, and B. Falsafi. Database servers on chip multiprocessors: Limitations and opportunities. In *CIDR 2007*, pages 79–87, 2007.

[11] T. Hey, S. Tansley, and K. M. Tolle. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. 2009.

[12] L. Hyafil and R. L. Rivest. *Graph partitioning and constructing optimal decision trees are polynomial complete problems*. 1973.

[13] R. Jin, N. Ruan, S. Dey, and J. X. Yu. SCARAB: scaling reachability computation on large graphs. In *SIGMOD 2012*, pages 169–180, 2012.

[14] G. Karypis and V. Kumar. Multilevel graph partitioning schemes. In *ICPP 1995*, pages 113–122, 1995.

[15] G. Karypis and V. Kumar. Parallel multilevel graph partitioning. In *IPPS 1996*, pages 314–319, 1996.

[16] G. Karypis and V. Kumar. A coarse-grain parallel formulation of multilevel k-way graph partitioning algorithm. In *PPSC 1997*, 1997.

[17] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Computing*, 20(1):359–392, 1998.

[18] T. Kiefer, B. Schlegel, and W. Lehner. Experimental evaluation of NUMA effects on database management systems. In *BTW 2013*, pages 185–204, 2013.

[19] S. Kintali. Betweenness centrality : Algorithms and lower bounds. *CoRR*, abs/0809.1906, 2008.

[20] T. Kissinger, T. Kiefer, B. Schlegel, D. Habich, D. Molka, and W. Lehner. ERIS: A numa-aware in-memory storage engine for analytical workload. In *ADMS 2014*, pages 74–85, 2014.

[21] J. Leskovec. Snap – stanford network analysis platform. http://snap.stanford.edu/snap/ [Online, last acessed 2016-02-30].

[22] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, pages 498–516, 1973.

[23] I. Pandis, R. Johnson, N. Hardavellas, and A. Ailamaki. Data-oriented transaction execution. *PVLDB*, 3(1):928–939, 2010.

[24] C. Rother, V. Kolmogorov, and A. Blake. "grabcut": interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, 2004.

[25] K. Schloegel, G. Karypis, and V. Kumar. *Graph partitioning for high performance scientific simulations*. 2000.

[26] J. Seo, J. Park, J. Shin, and M. S. Lam. Distributed socialite: A datalog-based language for large-scale graph analysis. *PVLDB*, 6(14):1906–1917, 2013.

[27] G. M. Slota, S. Rajamanickam, and K. Madduri. BFS and coloring-based parallel algorithms for strongly connected components and related problems. In *IEEE 2014*, pages 550–559, 2014.

[28] R. W. Taylor and R. L. Frank. CODASYL data-base management systems. *ACM Comput. Surv.*, 8(1):67–103, 1976.

[29] C. Walshaw, M. Cross, and M. G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *J. Parallel Distrib. Comput.*, 47(2):102–108, 1997.