Data Locality in Graph Databases through N-Body Simulation

Dominic Pacher
Institute of Computer Science
Technikerstrasse 21a
Innsbruck Austria
dominic.pacher@uibk.ac.at

Robert Binna
Institute of Computer Science
Technikerstrasse 21a
Innsbruck Austria
robert.binna@uibk.ac.at

Günther Specht
Institute of Computer Science
Technikerstrasse 21a
Innsbruck Austria
guenther.specht@uibk.ac.at

ABSTRACT

Data locality poses a major performance requirement in graph databases, since it forms a basis for efficient caching and distribution. This vision paper presents a new approach to satisfy this requirement through n-body simulation. We describe our solution in detail and provide a theoretically complexity estimation of our method. To prove our concept, we conducted an evaluation using the DBpedia dataset data. The results are promising and show that n-body simulation is capable to improve data locality in graph databases significantly.

Categories and Subject Descriptors

H.2.4 [**Database Systems**]: Graph databases

General Terms

Locality, N-body Simulation, Graph Data, Experimentation

Keywords

Database, Graph, Simulation, Graph Database, Triple Store

1. INTRODUCTION

Recently the demand to manage high amounts of linked data increased substantially. This development has its origin in data, generated by social as well as linked knowledge networks like Wikipedia [1]. In addition, all of today's imperative programming languages work on graph oriented (object) memory systems, because they are easy to understand and can be efficiently processed in main memory. Moreover, graph oriented memory systems provide means to easily formulate complex recursive behavior and data structures. Usually these data structures need to be stored persistently in some kind of external database.
Beside the exact internal concept, this (graph) database has to support query, update and remove operations of single nodes or complete sub graphs as fast as possible. Clearly this requirement influences all of the different sub components of a graph database and can be fulfilled through improvements on many different levels. However, there is no other property, which has as much influence on the performance and scalability of the overall system as *data locality*. In terms of graphs this means that any node stored has to be also physical near to its linked nodes in the memory. This seems to be a straightforward requirement, but it's hard to fulfill practically. In theory, a graph describes a multidimensional data structure, which has to be managed by the computer. Unfortunately, since memory systems work on a fixed one-dimensional memory layout, this cannot be done directly. The common solution to this problem is to define a mapping from multidimensional data to less (one) dimensional space. Although it's not a problem to find any kind of mapping, it's hard to preserve data locality at the same time. Therefore data locality isn't assured directly (Figure 1) and databases try to speed up operations using additional indexes or in the case of main memory systems, by providing cheap jumps through random access memory.

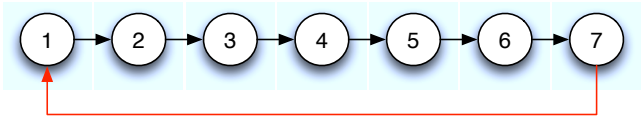

Figure 1: A two dimensional structure gets mapped to one-dimensional space. Since no locality is preserved, large jumps (6 nodes) appear in the data (red).

Despite the fact that this solutions work out quite well for the problem, they are always tied to additional costs and remaining limitations and don't solve the actual problem. For example, additional indices need space and have to be updated on every change. Main memory systems work well on one core and one computer. But since, frequent jumps between the cores memory or even worse, between computers, are orders of magnitudes costlier than jumps within main memory of one single thread, it's hard to distribute them properly.

To come up with a new approach to improve this situation, this paper suggests building a graph database whose nodes are aligned in memory by a n-body simulation system. Inspired by real world physics laws, links will be simulated as springs causing nodes to arrange themselves automatically. As a result, when the state of lowest energy is reached, a

23^{rd} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 31.05.2011 - 03.06.2011, Obergurgl, Austria.

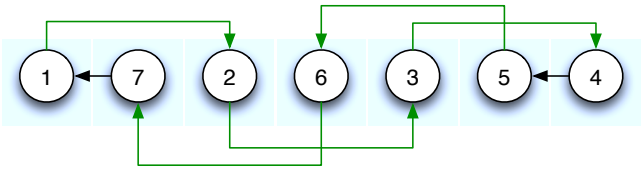

Figure 2: Better solution of Figure 1. Locality is preserved and the maximum jump length is reduced to two nodes (green)

maximum of data locality is provided at the same time (Figure 2). In addition, n-body simulation systems are known to be highly distributable and computational feasible [16].

Consequently the aim of this paper is to show through experimentation, that such a simulation will optimally place graph nodes in memory achieving improved data locality on global scale.

The remainder of this paper is structured as follows. Section 2 describes related papers in more detail. In section 3 we present our new method by a short introduction to n-body simulation idea, as well as some adjustments we had to make. To prove that our concept is feasible, we performed some preliminary evaluations which results are discussed in section 4. Section 5 sums up with an conclusion and future works.

1.1 Related Work

Although there is, at best of our knowledge, no related approach solving the data locality problem of graph database using n-body simulation, papers exists which make use of this method for related problems.

The idea of n-body simulation to support graph alignment has been already proposed in the 80s [14] and constantly improved [19]. However, these algorithms try to find an optimal layout for graph nodes, which is a far more complicated problem than preserving locality as it includes additional requirements like finding *aesthetic pleasing* solutions. Fortunately, this is clearly not an affordance for graph databases.

Plenty of systems were developed in the RDF research area trying to optimize storing and querying graphs. These *graph stores* can be separated into three groups of stores, which

- reside completely in memory (In-Memory Store)
- are based on a relational database (Relational Triple Store)
- use their own implementation (Native Triple Store)

To the group of *In-Memory Stores* belong GRIN [17] and Brahms [12], which mainly try to solve special purpose queries through dedicated indices. Also SpiderStore [6] operates in memory completely. However it makes no special assumptions about queries.

Jena SDB [18] and Virtuoso RDF Views [10] are part of the second group using a traditional row oriented *relational model*. Mapping graph data to the relational model tend to result in one big table with three columns: source node, edge, destination node (or in RDF terms subject, predicate object) and billions of rows. As the mapping of this table to a common row oriented store is inefficient [2] and [15] applied a column oriented relational model.

Part of the third group, the *native implementations*, are the adapted Jena TDB [18] (in contrast to SDB), Virtuoso [10], YARS [11] and RDF-3X [13]. Where the last two approaches make excessive use of indices to speed up the query execution process. Though RDF-3X achieved new query speed records, this approach is heavily optimized on read operations. As a consequence on any data update all indices have to be changed accordingly. In contrast, BitMap [3] uses a completely different design using compressed bit-matrix structure.

Finally Sesame [8] provides storage engines of all tree groups. The performance of these systems have been evaluated in [13] [6] and through the Berlin Benchmark [7].

Consequently there is no system yet using n-body simulation to improve data locality and it's interesting if such an approach is able to improve the overall performance of graph databases.

2. THE METHOD

In contrast to existing methods to store graph data we suggest an algorithm, which achieves a high degree of data locality. This algorithm is based on the idea, that link length don't come for free, making longer links to more distant data locations more expensive than shorter links. With this additional costing factor c, an optimal solution for the locality problem in databases can be defined as achieving the global minimum of the sum of this costs overall nodes n:

$$C_{all} = \min \sum_{i=0}^{n} c_i$$

This optimization process becomes quickly unsolvable using analytically methods, therefore a common n-body simulation approach is applied. Every edge is seen as a physical spring between two data nodes. Springs will add distance depended forces F_l to the connected links causing them to approach each other:

$$F_l = F_c * D(l)$$

Where F_c is the force constant and $D(l)$ a distance function of linked node l. This distance function can be for example a linear function returning the distance to the linked node l or an exponential function causing forces to increase exponentially with the distance.

Since a node is influenced by all its linked nodes, all forces F_l have to summed up to achieve the final overall force F_n:

$$F_n = \sum_{i=0}^{n} F_i$$

Now we can calculate the acceleration of the current node $n$using its mass m_n:

$$a_n = F_i / m_n$$

In our prototype we set m_n to 1 but for later implementations this parameter may represent a ideal way to reduce the movement of big nodes using the number of links as mass. This would cause big nodes to be moved less often. Finally we can use a_n to calculate the change of velocity

$$\Delta v_n = a_n * s$$

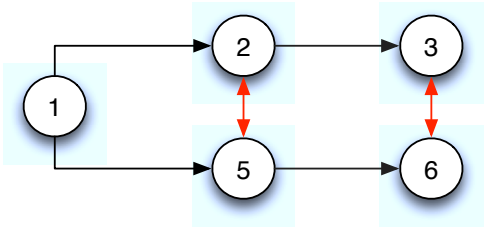

Figure 3: Nodes 2 / 5 and 3 / 6 have to be stored to the same position in one-dimensional space.

where s describes the used step size. For the sake of simplicity our prototype used a step size of 1. The simulation can now be formulated in three steps:

1. Calculate v_n for all n.
2. Change v_n according to Δv_n and calculate new position.
3. Check if there is any movement . If yes then goto 1.
4. Simulation finished.

2.1 Adjustments

N-body simulation methods have been used widely and very successfully in many fields of physics over the past decades. However some adjustments are necessary to make the approach useful for locality calculations.
As memory of all modern computers is accessed through discrete addresses, the simulation has to take this into account and have to operate on integers entirely. This approach has two advantages. In the first place it avoids the introduction of additional repulsion forces to keep nodes at a minimum distance to each other and secondly, the calculations can be done with faster integer calculations.
As mentioned previously, graph data is naturally multi dimensional, which stands in direct contrast to the one-dimensional memory space. Because of that, nodes may have found a final position, which is already be claimed by another node. Therefore, a priority function has to be defined to solve this problem, preserving that the node wins which leads to less energy in the overall system. This can be accomplished by using the nodes overall force as priority value.
An example of this problem can be found in Figures 3 and 4 where nodes 2/5 and 3/6 claiming the same position. Figure 4 also shows, that preserving locality comes at cost of the link length of other nodes.

2.2 Complexity Estimation

Generally the complexity of n-body simulations can be estimated as $\mathcal{O}(n * m * s_{num})$ where n describes the number of nodes, m the number of links per node and s the number of simulation steps needed until energy equilibrium. Consequently for a complete graph, where $n = m$ the complexity raises to $\mathcal{O}(n^2 * s_{num})$. This wouldn't be feasible for high amounts of data. Fortunately [4] showed for gravity simulations, which can be reduced to a fully connected graph, that complexity can be reduced to $\mathcal{O}(n * log(n) * s_{num})$, using a supporting tree structure and aggregation for distant nodes. Although this is the worst-case estimation, it is very unlikely to happen for real data where the number of links per node should always be significantly smaller than the number of

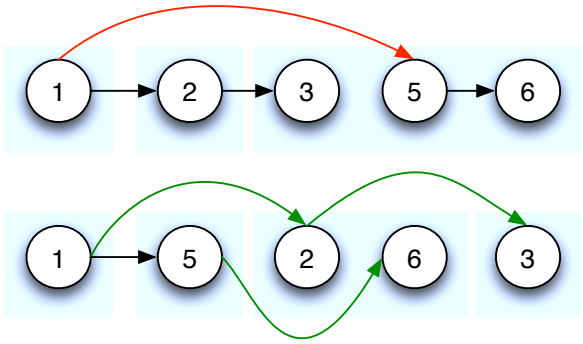

Figure 4: Mapping of Figure 3 to one dimension. The upper Figure shows a non optimal solution with jump size of three. The lower Figure shows a better solution with a jump size of two.

nodes. Indeed for a complete graph, the n-body simulation cannot improve any locality, because energy equilibrium is already reached and the simulation would terminate after the first step. Assuming that $n >> m$ the next important factor is s_{num}, which depends on the dataset as well as the used step size s. Consequently an increased step size would lead to less simulation steps. However too large steps sizes also increase the computational error between steps and will lead to an unstable simulation eventually. Fortunately, a variety of algorithms exist to minimize this approximation error, using one step or multistep methods as well as methods with variable step size at need [9].
Finally, if simulated once, we assume that the majority of data locations will remain stable and won't have to be recalculated on every data update on global scale. This estimation and in addition, that existing implementations deal with about 10 billion elements [16], let us believe that a large scale simulation of graph data is feasible.

3. PRELIMINARY EVALUATION

To prove the suggested concept we implemented a prototype and made some preliminary evaluations. To get realistic results we chose a subset (first 200 000 triples = 110,205 nodes) of the DBpedia dataset. All tests were conducted on a single machine (Mac Pro Intel Xeon 2,26 GHz) using a simple single threaded process with 200 MiB of dedicated ram.
To get an visual impression how effective our method increases data locality, we visualize every data element as a pixel in an image. As we are working on a one-dimensional space, all values are simply wrapped at the end of image width to create a two dimensional image. Every pixel position corresponds to the actual position of a data node in the data space.
In Figure 5 the color of this pixel represents the maximum distance of a node to its linked neighbor nodes in the data space. Using a maximum value of $n/2$ (value red) this Figure shows the development over time until energy equilibrium is reached. At $t = 0$ the data is scattered randomly in the

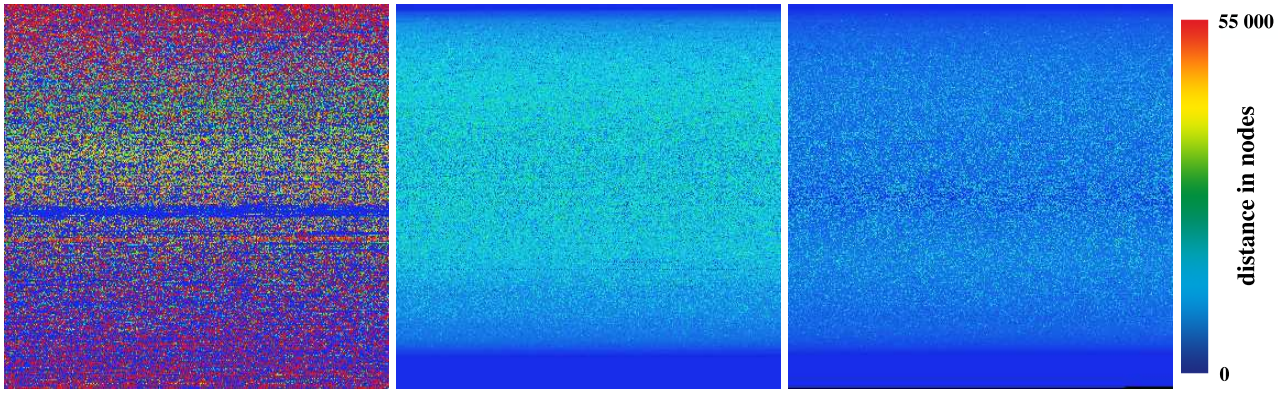

Figure 5: Complete data space of 110205 nodes. Each pixel represents one node. The color key on the right describes the maximum distance of a node's link. From left to right the image shows the data space at at t=0, t = 0.5 and t = 1.

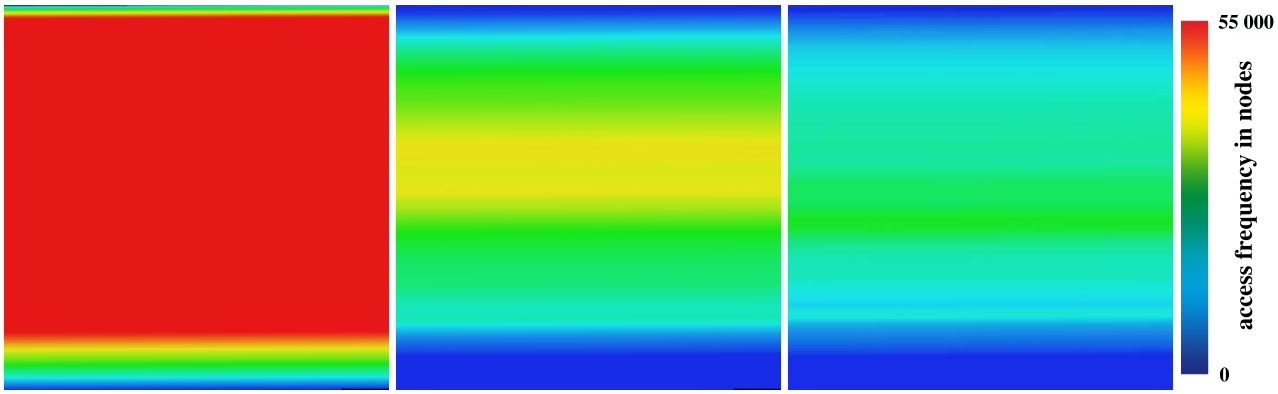

Figure 6: Series of access heat maps showing high access zones which should not be separated. Red positions are accessed by 55,000, dark blue nodes by less than 1,000 nodes. From left to right the image shows the data space at at t=0, t = 0.5 and t = 1.

data space. There are plenty of nodes, which have to jump through the whole data space to access their linked nodes. At $t = 0.5$ the data distribution has improved already, but can be further enhanced until a state of minimum energy (no more movement of nodes) is reached ($t = 1$). The final result shows that all nodes have now arrived at a position, where they can access their most distant linked node with a minimum of locality change.

To get a better impression of the exact numbers, we created a histogram (Figure 7) for the same sample data. For better understanding, be aware of the logarithmic scaling on the vertical axis. Furthermore we moved the mid-term frame $t = 0.5$ to quarter time $t = 0.25$ to get a better impression of the progress over time. Similar to Figure 5, at $t = 0$ one can observe an almost equal distribution of nodes along the complete range of possible distances. As an important matter of fact, there is already a peak of values having very close linked nodes on the very left side ($distance < 1,000$) of the diagram. Since it's often the case that new nodes are introduced followed by their direct neighbors, the input file itself can be seen as origin of this peak. Of course this issue can only have a positive impact on local data locality and not on global scale, as we want to achieve. During the simulation over $t = 0.25$ to $t = 1.0$ one can observe a significant reduction of distances to about one quarter of the originally data space. In addition, the histogram points out that global data locality comes also at cost of local data locality, which is showed by the reduction of the red peak at $t = 0$ mentioned before to the blue one at $t = 1.0$. The cause of this reduction can be seen again in the mapping of multi dimensional data to less (in our case one) dimensional space, where different data nodes claim the same position (Figures 3 and 4).

Furthermore there are some small peaks (two nodes) remaining near distance $50,000$. These nodes couldn't be aligned very well. Although we always expected problems with nodes that are highly linked to different other nodes and can't be further optimized by means of location, this appears not to be the case here, as the number of links of the worst-case node where only about 300. Unfortunately, this remains a rather unsatisfied situation and has to be further investigated in future. However, as shown in our sample data set, these nodes can be considered as very rare (overall 8 nodes out of $110,025$ until distance of $26,000$).

These problems apart, it's most important that the global data locality improved substantially. As previously seen in Figure 5, the histogram shows that most nodes are far less distant to their linked nodes than at the start of simulation.

Figure 7: Histogram of the occurrence of nodes and their maximum distances to linked nodes. For better understanding the time frames were changed to t=0, t = 0.25 and t = 1.

In particular, the link length was reduced to 1/4 in the worst and to 1/10 in the average case.
Based on this data it should be possible to make efficient decisions how graph data can be separated in certain chunks, to be distributed on different cores as well as on different computers. To gather more insight into this, we used an access heat map. To create this map all data positions lying between a node and all it's linked nodes are incremented by one. Of course in main memory we are able to randomly access every position at same speed, but in a distributed environment this model fits very well. When this is done for all nodes and their respective linked nodes, every position marks the number of accesses needed to visit every neighbor node within data space (Figure 6). This image gives an impression, where the data space can be separated best, choosing less dense (blue in the Figure) zones.

4. CONCLUSION AND FUTURE WORK

The aim of this paper was to show that a n-body simulation can improve graph data locality significantly. After a introduction to our suggested method, we evaluated a experimental prototype using partial data of the DBpedia dataset [5]. As a result, we were able to restrict jumps to about 1/4 of the whole data space in the worst-case and to 1/10 for the average case. Although these are very promising results, there is plenty of work remaining.
We theoretically showed that our n-body approach should scale well into millions of graph nodes. However, our prototype is currently not optimized for very large data sets like the complete DBpedia dataset, consisting of about 100 million triples. Hence our goal for future works will be to optimize the simulation by improving the algorithm and finding a way to distribute the simulation on many cores and computers. As a result of this development, we hope to provide practically evidence that our method is working on large real world graphs preserving computational feasibility.

5. REFERENCES

[1] *Wikipedia Free Encyclopedia*. http://wikipedia.com, apr 2011.
[2] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. *Scalable semantic web data management using vertical partitioning*. VLDB Endowment, sep 2007.
[3] M. Atre, V. Chaoji, M. J. Zaki, and J. A. Hendler. *Matrix Bit loaded: a scalable lightweight join query processor for RDF data*. ACM, apr 2010.
[4] J. Barnes and P. Hut. A hierarchical O(N*log(N)) force-calculation algorithm. *nature*, 324(4):446–449, 1986.
[5] C. Becker. *RDF Store Benchmarks with DBpedia*. www4.wiwiss.fu-berlin.de, 2011.
[6] R. Binna, W. Gassler, E. Zangerle, and D. Pacher. SpiderStore: Exploiting Main Memory for Efficient RDF Graph Representation and Fast Querying. 2010.
[7] C. Bizer. The berlin sparql benchmark. *Int J Semantic Web Inf Syst*, 2009.
[8] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In *Semantic Web - Iswc 2002*, pages 54–68, Aidministrator Nederland BV, Amersfoort, Netherlands, 2002. Aidministrator Nederland BV, Amersfoort, Netherlands.
[9] J. Butcher. *Numerical Methods for Ordinary Differential Equations* . Wiley, 2 edition, jun 2008.
[10] O. Erling and I. Mikhailov. Rdf support in the virtuoso dbms. In T. Pellegrini, S. Auer, K. Tochtermann, and S. Schaffert, editors, *Networked Knowledge - Networked Media*, volume 221 of *Studies in Computational Intelligence*, pages 7–24. Springer Berlin / Heidelberg, 2009.
[11] A. Harth, J. Umbrich, and A. Hogan. YARS2: A federated repository for querying graph structured

data from the web. *The Semantic Web*, 2007.

[12] M. Janik and K. Kochut. BRAHMS: A WorkBench RDF Store and High Performance Memory System for Semantic Association Discovery. In *Fourth International Semantic Web Conference*, pages 431–445. Springer, 2005.

[13] T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *The VLDB Journal — The International Journal on Very Large Data Bases*, 19(1):91–113, feb 2010.

[14] E. Peter. A Heuristic for Graph Drawing. *Congressus Numerantium*, 42:149–160, nov 1984.

[15] L. Sidirourgos, R. Goncalves, M. Kersten, N. Nes, and S. Manegold. Column-store support for rdf data management: not all swans are white. *Proc. VLDB Endow.*, 1:1553–1563, August 2008.

[16] V. Springel, S. D. M. White, A. Jenkins, C. S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, J. A. Peacock, S. Cole, P. Thomas, H. Couchman, A. Evrard, J. o. r. Colberg, and F. Pearce. Simulations of the formation, evolution and clustering of galaxies and quasars. *nature*, 435(7042):629–636, jun 2005.

[17] O. Udrea, A. Pugliese, and V. S. Subrahmanian. Grin: a graph based rdf index. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, pages 1465–1470. AAAI Press, 2007.

[18] K. Wilkinson, C. Sayers, and H. Kuno. Efficient RDF storage and retrieval in Jena2. In *Proceedings of SWDB*, 2003.

[19] V. Zabinako and P. Rusakovs. Development and Implementation of Partial Hybrid Algorithm for Graphs Visualization. *Scientific Proceedings of Riga Technical University*, 5(34):192–203, jul 2008.