TECHNISCHE UNIVERSITÄT DRESDEN

Allocation Strategies for Data-Oriented Architectures

Kurzfassung der Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
Dipl.-Inf. Dipl.-Math. Tim Kiefer
geboren am 25. September 1983 in Saalfeld

Betreuender Hochschullehrer:
Prof. Dr.-Ing. Wolfgang Lehner

Dresden, Juli 2015

Allocation Strategies in Data-Oriented Architectures (Extended Abstract)

Tim Kiefer

Data orientation is a common design principle in distributed data management systems. In contrast to process-oriented or transaction-oriented system designs, data-oriented architectures are based on data locality and function shipping. Data-oriented systems, i.e., systems that implement a data-oriented architecture, bundle data and operations together in tasks which are processed locally on the nodes of the distributed system. Allocation strategies map tasks to nodes and are core components in data-oriented systems. Optimal allocation strategies are hard to find given the complexity of the systems, the complicated interactions of tasks, and the huge solution space.

In this thesis, we develop novel allocation strategies for data-oriented systems based on graph partitioning algorithms. We propose to extend classic graph partitioning to model non-linear performance by introducing vertex weights that do not linearly aggregate to partition weights. On top of the basic algorithms, we propose methods to incorporate heterogeneous infrastructures and to react to changing workloads and infrastructures by incrementally updating the partitioning.

We evaluate all components of our allocation strategy algorithms and show their applicability and scalability with synthetic workload graphs. In end-to-end–performance experiments in two actual data-oriented systems, a database-as-a-service system and a database management system for multiprocessor systems, we prove that our allocation strategies outperform alternative state-of-the-art methods.

1 Introduction

The term *data-oriented architecture* is not fixed and does not refer to a single system layout. In the context of data management systems, data orientation is commonly used to describe distributed data processing systems that make the data a first class citizen and drive the processing based on data locality. Unlike process-oriented or transaction-oriented architectures, where a single process is assigned to acquire the necessary data, perform the operations, and return the results, in data-orientated architectures the data drives the processing. In data-oriented systems, i.e., systems

that implement a data-oriented architecture, transactions[1] are split into self contained units of work. Data and the operations performed thereon are bundled in *tasks* and tasks are processed locally on the *nodes* of the distributed system. The term *allocation* in data-oriented systems refers to the mapping of tasks to nodes. Data locality is a key design principle in data-oriented systems, i.e., tasks are executed where the data resides instead of moving the data to the process. In other words, a data-oriented architecture uses function shipping while process-oriented architectures use data shipping. Data can be moved in data-oriented systems, but only explicitly when tasks communicate to coordinate or perform complex workloads.

In this thesis, we investigate allocation strategies for systems that implement data-oriented architectures. We demonstrate the applicability of our solutions in two different exemplary systems from different application domains, namely a database-as-a-service system and a database management system (DBMS) for multiprocessor systems. Although the two scenarios are very different, they base on common principles and we show that a unified allocation strategy can be used for both.

Given that the location of data determines where processing is performed, the allocation is crucial in any data-oriented system. Consolidation, i.e., co-location and concurrent execution of (possibly unrelated) tasks on single nodes is a central means to optimize utilization, hence cost-effectiveness, and performance in data-oriented systems (Curino et al., 2011). Consequently, the decision on where data resides in data-oriented systems, i.e., the allocation strategy, is commonly taken from the user and optimized by the system. This optimization is workload-driven, i.e., based on the actual tasks. An ideal allocation strategy can be used to optimize for different objectives like minimizing the required number of nodes or maximizing task performance.

Finding an optimal allocation is an inherently difficult problem. Even on a small scale, solving the allocation problem is hard, given the complex interactions that tasks may have. Contention on hardware or operating system resources may lead to performance characteristics that are hard to predict. The presence of heterogeneous hardware, dynamic workloads, and dynamic infrastructures add to the complexity of the problem. To ensure the scalability of data-oriented systems and to keep them manageable with hundreds of thousands of tasks and thousands of nodes, fast and reliable algorithms for the allocation problem are mandatory. However, distributed data processing systems and complex workloads are hard to model and system behavior is hard to predict (Ahmad and Bowman, 2011; Curino et al., 2011). Even with the models, the allocation problem has a huge solution space and the problem itself is NP-hard (e.g., Curino et al., 2011; Schaffner et al., 2013).

The core contribution of this thesis is a set of allocation strategies for data-oriented systems. Our allocation strategies model workload and infrastructure information in weighted graphs and solve the allocation problem based on graph partitioning and mapping algorithms. We extend known heuristics for graph partitioning by methods to account for non-linear resource behavior. Thereby, we introduce the notion of *penalized weights* and *secondary weights*. To the best of our knowledge, our method is the first to partition a graph with vertex weights that do not combine linearly to partition weights.

[1]The term *transaction* here refers to any query, statement, or data-processing code

2 Foundations of Data-Oriented Systems

As a foundation for our allocation strategies, we present two application scenarios that use data-oriented systems, namely *database-as-a-service systems* and *DBMSs for multiprocessor systems*. A variety of actual systems exist in both scenarios that differ in the implementation or in the degree of data orientation, i.e., the granularity of tasks and the degree of data locality. Although being different, we show in our thesis that all systems can use a unified allocation strategy once the workload and the infrastructure are abstracted in the corresponding models.

2.1 Data Orientation in Database-as-a-Service Systems

Cloud computing has become a valid usage model for a variety of applications and a successful business model for several service providers. Offering (relational) databases as a service transfers the advantages of cloud computing to the data storage layer and allows applications that rely on a storage layer to run in the cloud.

Database-as-a-service systems implement data-oriented architectures. Relational databases comprise the actual data (in form of tables) and the operations thereon (in form of SQL statements). Relational databases can be partitioned, leading to tasks that communicate to execute a workload. The allocation problem is of high interest in database-as-a-service systems (e.g., Curino et al., 2011; Schaffner et al., 2013). The allocation strategy can, for instance, be used to minimize the number of nodes and thereby reduce resource consumption and operational costs. Furthermore, the allocation strategy can help to improve performance, to guarantee service levels, and to ensure availability.

Database-as-a-service systems are implemented by virtualizing the database server and consolidating multiple virtual servers on a single physical server. However, the multi-layered system stack of a database management system, ranging from the hardware layer up to the database schema, allows for virtualization of different levels

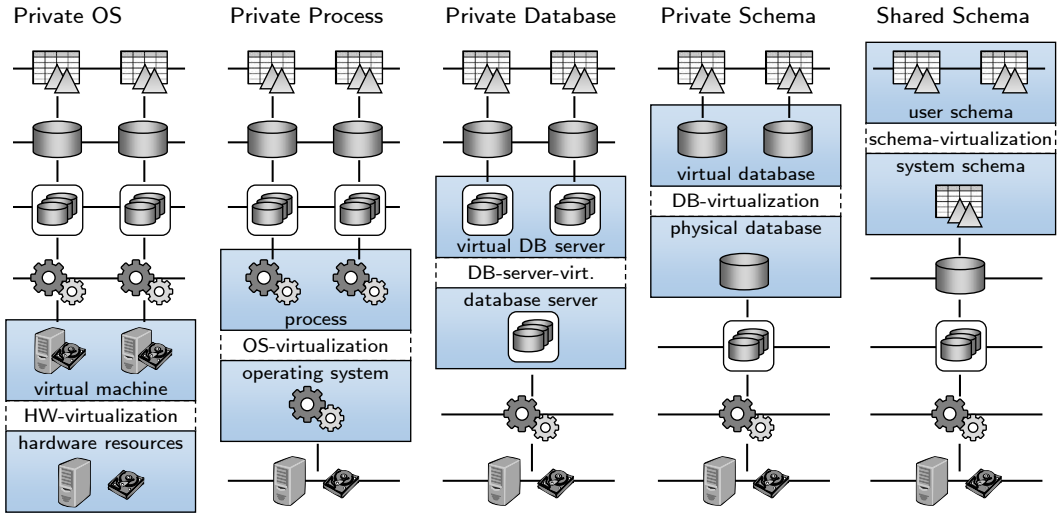

Figure 1: Classification of Database-as-a-Service Systems

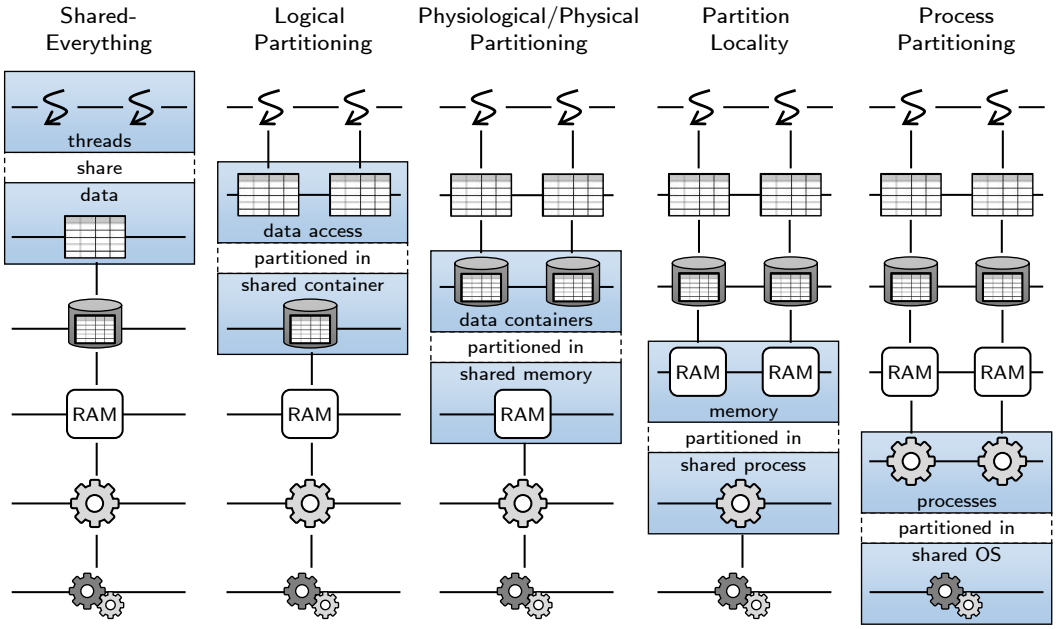

Figure 2: Classification of Data-Oriented Approaches to Modern Hardware

with different characteristics of the resulting system. This leads to the following five classes, which are also shown in Figure 1: (1) Private OS, (2) Private Process, (3) Private Database, (4) Private Schema, and (5) Shared Schema (e.g., Jacobs and Aulbach, 2007; Kiefer and Lehner, 2011; Wang et al., 2008).

The data-oriented systems differ among the implementation classes. Especially, the granularity of tasks and nodes depends on the layer that is virtualized and consequently on the portion of the system stack that is shared/private.

2.2 Data Orientation in DBMSs for Multiprocessor Systems

The second category of data-oriented systems considered in our thesis are DBMSs for modern multiprocessor systems. Classic DBMSs (not being optimized for these systems) contain many critical sections and central data structures that quickly lead to contention and hinder scalability (e.g., Huber and Freytag, 2009; Johnson et al., 2008, 2009; Pandis et al., 2010; Salomie et al., 2011).

Database management systems can leverage the data-oriented–execution principle an treat multiprocessor machines as distributed systems. Relations are partitioned and partitions together with the operations thereon form tasks. Tasks are mapped to multiprocessors of the machine, i.e., the nodes of the data-oriented system.

Different approaches have been proposed to deal with the characteristics of modern hardware in database management systems. We propose a classification (shown in Figure 2) that distinguishes classes based on the step in the data access stack where the data or the access is partitioned. This classification scheme leads to the following five classes: (1) Shared-Everything, (2) Logical Partitioning, (3) Physiological/Physical Partitioning, (4) Partition Locality, and (5) Process Partitioning.

3 Allocation Problem for Data-Oriented Systems

In this section, we introduce the allocation problem for data-oriented systems. To use the allocation strategy, specific data-oriented systems have to be generalized to an abstract data-oriented system. The abstract system is a common denominator of different data-oriented systems and acts as an interface to the allocation strategy. Figure 3 sketches exemplarily the generalization of a (class of) database-as-a-service system(s) to the abstract data-oriented system.

3.1 Infrastructure and Workload Model

Characteristics of the execution environment are subsumed in the *infrastructure*. The infrastructure contains hardware resources that are captured, their types (bounded or unbounded) and their models to combine loads (linear or non-linear). Properties of the tasks in the data-oriented system are subsumed in the *workload*.

Definition 1 (Infrastructure)**.** An *infrastructure* is an undirected graph of *nodes* connected by *links*. Nodes have bounded and unbounded resources and weight functions map each node to either an absolute or a relative capacity per resource. Each unbounded resource additionally has a model for combining loads. Link capacities in the infrastructure are given by an edge-weight function.

Although not explicitly stated, the infrastructure can be heterogeneous. Different kinds of actual hardware resources can be modeled in the infrastructure, e.g., processing resources (CPU cycles), memory resources (bandwidth, capacity), and network resources (link bandwidth). The allocation strategy finds mappings based on the abstract infrastructure instead of any specific resource.

The infrastructure model distinguishes between *bounded resources* and *unbounded resources*. Bounded resources have a hard limit that cannot be exceeded. Overloading a node's bounded resources leads to an invalid allocation. Unbounded resources

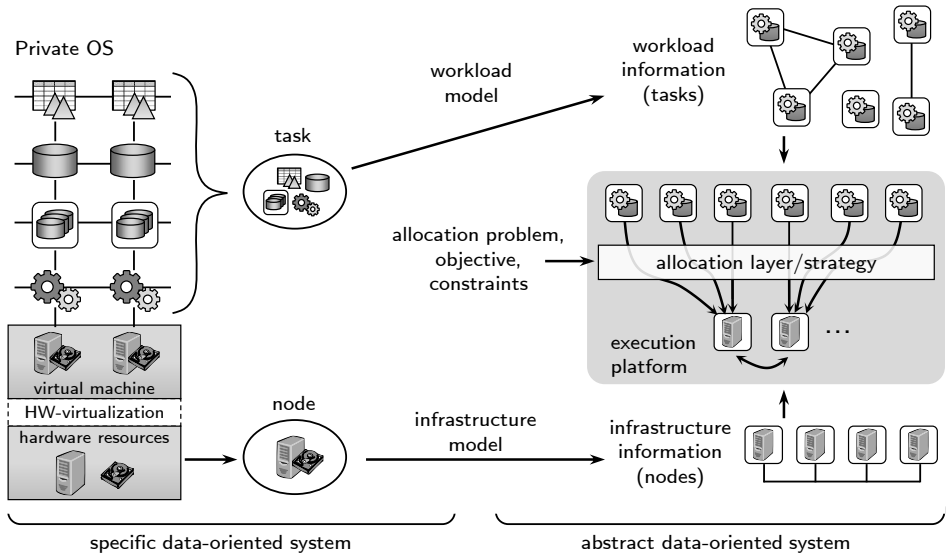

Figure 3: Abstracting a Specific Data-Oriented System

are not literally unbounded but they can (and usually will) be overcommited. Overcommitting an unbounded resource still leads to a valid allocation, even if the performance degrades. The goal of our allocation strategy will be to respect the upper bounds of bounded resources and to balance unbounded resources.

Several tasks commonly share a single node in data-oriented systems. To be able to evaluate a given allocation, the aggregated load of a node (induced by all tasks) needs to be estimated. To grasp the behavior of complex systems, we assume a simplified resource consumption model that reflects non-linear performance often found in actual systems. In our infrastructure model, each resource has a (relative) capacity and a (possibly non-linear) model to combine loads.

Definition 2 (Workload)**.** A *workload* is an undirected graph of *data partitions* connected by *data transfers*. Data partitions and operations executed on them are bundled in *tasks*. Tasks consume bounded and unbounded resources and vertex weight functions map each task to either an absolute or a relative cost, i.e., load, per resource. An edge weight function quantifies data transfer costs.

The granularity of a data partition in the workload depends on the implementation of the data-oriented system, e.g., it may be a single tuple, a relation or parts thereof, or a whole database. Likewise, a task is a generalization of a query, a transaction, or any complex set of operations on the data. Data-oriented systems may look differently and it heavily depends on the actual system how the workload graph can be derived from the performed tasks. In fact, the workload does not even have to be in the form of SQL statements. A program, written in a high-level language, that contains data manipulating operations may as well be transformed to a workload.

3.2 Allocation Problem

The allocation problem can be formulated differently to optimize for different objectives. Two basic formulations are (1) minimize the number of used resources without violating performance constraints and (2) maximize performance with a given amount of resource. Both formulations can be extended and customized to further specify the allocation problem.

Given the complexity of the underlying systems, performance may be hard to evaluate without actually materializing a setup and executing the workload. Instead, meta-objectives that can be evaluated are used in practice to achieve the actual goals. For instance, the Schism project (Curino et al., 2010) uses the number of distributed transactions, i.e., the message count, as a meta-objective. Minimizing the number of transactions that have to communicate is shown to improve performance.

Our allocation strategies optimize performance by minimizing costly communication and balancing oversubscribed unbounded resources. The first assumption is that network communication is a major cost factor that, given a partitioning, can be optimized by the allocation strategy. The second assumption is that balancing load, especially for oversubscribed resources, leads to the best global performance.

Given these preconditions, we use the following definition for an *allocation* and a corresponding formulation for the *allocation problem* to solve.

Definition 3 (Allocation)**.** An *allocation* is a mapping from workload tasks to infrastructure nodes. For each resource, a node's load is the combined weight (including non-linear behavior) of all tasks assigned to that node. An allocation is *valid* if no load is greater than the node's capacity for all bounded resources. An allocation is *balanced* if all loads are equal (within a tolerance) to the average load for all unbounded resources. An allocation's *communication costs* are the sum of all edge weights of edges between vertices that are assigned to different nodes.

Allocation Problem

Given a workload and an infrastructure, the allocation problem is to find a valid and balanced allocation that minimizes communication costs.

A number of requirements can be derived for allocation strategies. The infrastructure model assumes that performance does not scale linearly with the amount of work. The allocation strategy must therefore be able to handle this non-linear infrastructure behavior. Furthermore, the allocation strategy must consider heterogeneous infrastructures, i.e., nodes with different capacities, network graphs that are not fully connected, and links with different capacities. Multiple bounded and unbounded resources as well as communication costs must be considered individually by the allocation strategy. The allocation strategy must be able to incrementally update a solution after changes in the workload or the infrastructure occurred.

4 Balanced K-Way Min-Cut Graph Partitioning

In this section, we describe the balanced k-way min-cut graph partitioning problem (GPP) and methods to solve it. The general goal of the problem is to partition a given graph into k parts such that the sum of edges that are cut is minimized while keeping the sizes of all parts within balance. Applied to the workload graph, a balanced min-cut partitioning minimizes communication and at the same time balances load across the nodes.

To develop an allocation strategy, we start with simplified formulations of infrastructure, workload, and allocation. In the GPP, we assume a homogeneous infrastructure with only a single unbounded and linear resource. These limitations will be relaxed step-by-step. Multiple individual resources are covered in the Multi-Constraint GPP (MC-GPP).[2] The Penalized GPP (P-GPP) and the Secondary Weight GPP (SW-GPP)[3] extend the graph partitioning algorithms to reflect non-linear resources. Extensions to the core graph partitioning algorithms further relax limitations of the allocation strategy. We propose methods to incrementally update an allocation, deal with heterogeneous infrastructures, and incorporate bounded resources.

[2]Due to space constraints, the MC-GPP is omitted in this extended abstract.
[3]The SW-GPP is also omitted in this extended abstract.

4.1 Graph Partitioning

Given an undirected, weighted graph, the balanced k-way min-cut graph partitioning problem (GPP) refers to finding a k-way partitioning of the graph such that the total edge cut is minimized and the partitions are balanced within a given tolerance. Minimizing the edge cut can be considered the objective in the GPP while the requirement that partitions are of similar size can be considered the constraint.

Partitioning a graph into k partitions of roughly equal size such that the total cut is minimized is NP-complete (Hyafil and Rivest, 1973). There exist approximation algorithms with proven boundaries for variations of the problem. However, these results and the implementations of the methods are only of theoretical interest due to extremely long runtimes (Andreev and Racke, 2004; Buluç et al., 2013). Heuristics, such as the multilevel graph partitioning framework, are used in practice to solve the GPP (Bichot and Siarry, 2011; Buluç et al., 2013).

The multilevel graph partitioning framework consists of three phases: (1) *coarsening* the graph, (2) finding an *initial partitioning* of the coarse graph, and (3) *uncoarsening* the graph and projecting the coarse solution to the finer graphs.

The idea of multilevel graph partitioning is to reduce the initial problem to a much smaller problem, then to solve the small problem, and last to project the solution of the small problem to the initial problem. To improve the quality of the final solution, each uncoarsening step is followed by a refinement step. There are several intuitive reasons why the multilevel approach leads to very good results. First, one can use very expensive algorithms or a variety of algorithms on the coarse level without increasing the overall execution time by a lot. Second, coarsening broadens the scope of local optimization algorithms, i.e., moving vertices between partitions in the coarse graph leads to potentially long distance moves of many vertices in the finest graph. Third, local improvements during the uncoarsening are fast because they start with an already good solutions.

4.2 Penalized Graph Partitioning

There is a mismatch between the graph partitioning problem and the actual behavior of the infrastructure. A fundamental assumption in the graph partitioning problem is that vertex weights sum up to reflect the weight of a partition. To account for the non-linear performance caused by contention in the actual infrastructure, we propose the *Penalized Graph Partitioning Problem* (P-GPP). The P-GPP is a generalization of the GPP where the goal is to find a k-way partitioning with minimal cut such that the *penalized partition weights* are balanced within a given tolerance.

Definition 4 (Penalized Weighted Graph)**.** Let $G = (V, E, w_V, w_E, p)$ be an undirected, weighted graph with vertexes V, edges E, a vertex weight function w_V, and an edge weight function w_E. Furthermore, let p be a positive, monotonic increasing penalty function that penalizes a partition weight based on the partition cardinality:

$$p\colon \mathbb{N} \to \mathbb{R}_{\geq 0} \text{ with } p(n_1) \leq p(n_2) \text{ for } n_1 \leq n_2.$$

The vertex weight function is extended to sets $V' \subseteq V$ (penalized partition weights):

$$w_V(V') := \sum_{v \in V'} w_V(v) + p(|V'|).$$

Adding penalties to partition weights invalidates some of the assumptions made in the GPP and its solution algorithms. Most fundamentally, the combined weight of two or more partitions is not equal to the weight of a partition containing all the vertices. Consequently, the *total vertex weight* $w_V(V)$ of a graph differs from the *total partition weight* of the same graph. The total partition weight is the sum of all partition weights. Consequently, the total partition weight depends on the partitioning and is not constant for a given graph. This observation has implications in all steps of the graph partitioning algorithm, e.g., the balance constraint needs to be modified to refer to the total partition weight instead of the total vertex weight. In our thesis, we propose modifications of the multilevel graph partitioning algorithm to solve the P-GPP.

Experimental Evaluation of Graph Partitioning In the thesis, we thoroughly evaluate graph partitioning as an allocation strategy. We show the benefit of multi-constraint and penalized graph partitioning over standard graph partitioning. We confirm the applicability of graph partitioning as a solution for the allocation problem by presenting results of various scalability experiments. More specifically, the graph partitioning algorithms scale well with the number of nodes, the number of edges, the number of partitions, and the number of constraints (i.e., vertex weights). Last, in the thesis, we evaluate our extensions of the graph partitioning algorithms that (1) incrementally update a partitioning, (2) deal with heterogeneous infrastructures, and (3) incorporate capacity constraints for bounded resources.

5 Experimental Evaluation

In our thesis, we experimentally evaluate our allocation strategies in two data-oriented systems. First, the experiments are used to test whether the allocation strategies are applicable in actual data-oriented systems. Second, the experiments help to evaluate the benefit of our new allocation strategies over existing strategies.

Our first test system is the fully functional database-as-a-service system MTM, which we developed as part of the thesis. We conduct our experiments using our own Multi-Tenancy Database Benchmark Framework MulTe (Kiefer et al., 2012) and Amazon Web Services[4]. Multiple EC2 instances are used as backend nodes. Additional EC2 instances are used to host the benchmark application, frontend tools, and the middleware. We deploy two different setups for our experiments. The larger setup uses 32 instances as backend nodes and 4 instances drive the benchmark. In this setup, we host a total of 1,000 databases with an aggregated size of 120 GiB.

Our second test system is the ERIS database management system for multiprocessor systems.[5] A defining characteristic of multiprocessor systems is the non-uniform memory access (NUMA) caused by the main memory being directly attached to the various sockets and hence distributed across the system. Differences in latency

[4] `http://aws.amazon.com`

[5] ERIS is a research prototype developed by the Dresden Database Systems Group at the TU Dresden (Kissinger et al., 2014).

and bandwidth up to a factor of 10 between local and remote memory accesses lead to the conclusion that NUMA systems should be treated as distributed systems. The ERIS system is an in-memory DBMS that is optimized for the challenges of multiprocessor systems with non-uniform memory access.

The experiments in both systems reveal that a simple first fit allocation strategy fails to balance the load, which leads to a skewed system and high relative response times. Comparing the allocation strategies that are based on graph partitioning, the experiments show that penalized graph partitioning leads to better overall system performance in all metrics. Penalized graph partitioning causes fewer outliers than the unmodified graph partitioning and has better maximum, average, and median relative response times.

6 Conclusion

Data orientation is a common design principle in distributed data management systems. The abstract data-oriented architecture, which is based on data locality, is implemented in different systems in a variety of application scenarios. Allocation strategies are core components in any data-oriented system. Good allocation strategies can lead to balanced systems while others cause skew in the load and therefore the application performance and the infrastructure utilization. Optimal allocation strategies are hard to find given the complexity of the systems, the complicated interactions of tasks, and the huge solution space.

We show in the thesis that data-oriented systems from different application scenarios and with different abstraction levels can be generalized to generic infrastructure and workload descriptions. We use weighted graph representations to model infrastructures with bounded and unbounded resources and possibly non-linear performance characteristics. Based on our workload and infrastructure model, we formalize the allocation problem, which seeks a valid and balanced allocation that minimizes communication costs.

Our unified allocation strategies for the generalized data-oriented system are based on the balanced k-way min-cut graph partitioning problem and the corresponding multilevel graph partitioning solution heuristic. We show methods to solve the graph partitioning problem for single and multiple resources and propose methods for resources with non-linear performance characteristics. On top of the basic algorithms, we propose extensions to (1) incorporate heterogeneous infrastructures, (2) react to changing workloads and infrastructures by incrementally updating a partitioning, and (3) enable bounded resources in the infrastructure model.

Experimental evaluations of all components of our allocation strategies on synthetic workload graphs confirm the applicability and scalability of the methods. In end-to-end–performance experiments in MTM and ERIS, we prove that our allocation strategies can be used in actual data-oriented systems and that they outperform alternative state-of-the-art allocation strategies.

References

Ahmad, Mumtaz and Ivan T. Bowman (2011). "Predicting System Performance for Multi-Tenant Database Workloads". In: *Proceedings of the 4th International Workshop on Testing Database Systems (DBTest '11)*. Athens, Greece. Link.

Andreev, Konstantin and Harald Racke (2004). "Balanced Graph Partitioning". In: *Proceedings of the 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'04)*. Barcelona, Spain, pp. 120–124. Link.

Bichot, Charles-Edmond and Patrick Siarry, eds. (2011). *Graph Partitioning*. Wiley.

Buluç, Aydin, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz (2013). "Recent Advances in Graph Partitioning". In: *preprint: Computing Research Repository*, pp. 1–37. arXiv: `arXiv:1311.3144v3`. Link.

Curino, Carlo, Evan P. C. Jones, Y. Zhang, and Sam Madden (2010). "Schism: a Workload-Driven Approach to Database Replication and Partitioning". In: *Proceedings of the 36th International Conference on Very Large Data Bases (VLDB '10)*. Singapore, China, pp. 48–57. Link.

Curino, Carlo, Evan P. C. Jones, Sam Madden, and Hari Balakrishnan (2011). "Workload-Aware Database Monitoring and Consolidation". In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11)*. Athens, Greece, pp. 313–324. Link.

Huber, Frank and Johann-Christoph Freytag (2009). "Query Processing on Multi-Core Architectures." In: *Grundlagen von Datenbanken*, pp. 27–31. Link.

Hyafil, Laurent and Ronald L. Rivest (1973). *Graph Partitioning and Constructing Optimal Decision Trees are Polynomial Complete Problems*. Tech. rep. IRIA – Laboratoire de Recherche en Informatique et Automatique. Link.

Jacobs, Dean and Stefan Aulbach (2007). "Ruminations on multi-tenant databases". In: *Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW '07)*. Aachen, Germany, pp. 5–9. Link.

Johnson, Ryan, Ippokratis Pandis, and Anastasia Ailamaki (2008). "Critical Sections Re-emerging Scalability Concerns for Database Storage Engines". In: *Proceedings of the Fourth International Workshop on Data Management on New Hardware (DaMoN '08)*. Vancouver, British Columbia, Canada, pp. 35–40. Link.

Johnson, Ryan, Ippokratis Pandis, Nikos Hardavellas, Anastasia Ailamaki, and Babak Falsafi (2009). "Shore-MT: A Scalable Storage Manager for the Multicore Era". In: *Proceedings of the 12th International Conference on Extending Database Technology (EDBT '09)*. Saint Petersburg, Russia, pp. 24–35. Link.

Kiefer, Tim and Wolfgang Lehner (2011). "Private Table Database Virtualization for DBaaS". In: *Proceedings of the 4th IEEE International Conference on Utility and Cloud Computing (UCC '11)*. Melbourne, Australia, pp. 328–329. Link.

Kiefer, Tim, Benjamin Schlegel, and Wolfgang Lehner (2012). "MulTe: A Multi-Tenancy Database Benchmark Framework". In: *Proceedings of the 4th TPC Technology Conference on Performance Evaluation & Benchmarking (TPCTC '12)*. Istanbul, Turkey. Link.

Kissinger, Thomas, Tim Kiefer, and Benjamin Schlegel (2014). "ERIS: A NUMA-Aware In-Memory Storage Engine for Analytical Workloads". In: *Proceedings of the 5th International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS '14)*. Hangzhou, China. Link.

Pandis, Ippokratis, Ryan Johnson, Nikos Hardavellas, and Anastasia Ailamaki (2010). "Data-Oriented Transaction Execution". In: *Proceedings of the 36th International Conference on Very Large Data Bases (VLDB '10)*. Singapore, China. Link.

Salomie, Tudor-Ioan, Ionut Emanuel Subasu, Jana Giceva, and Gustavo Alonso (2011). "Database Engines on Multicores, Why Parallelize When You Can Distribute?" In: *Proceedings of the 6th European Conference on Computer Systems (EuroSys '11)*. Salzburg, Austria, pp. 17–30. Link.

Schaffner, Jan, Tim Januschowski, Megan Kercher, Tim Kraska, Hasso Plattner, Michael J. Franklin, and Dean Jacobs (2013). "RTP: Robust Tenant Placement for Elastic In-Memory Database Clusters". In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*. New York, New York, USA, pp. 773–784. Link.

Wang, Zhi Hu, Chang Jie Guo, Bo Gao, Wei Sun, Zhen Zhang, and Wen Hao An (2008). "A Study and Performance Evaluation of the Multi-Tenant Data Tier Design Patterns for Service Oriented Computing". In: *Proceedings of the 2008 IEEE International Conference on e-Business Engineering (ICEBE '08)*. Xi'an, China, pp. 94–101. Link.