Mixed Workload Management for In-Memory Databases

Johannes Wust
supervised by Prof. Dr. Hasso Plattner
Hasso Plattner Institute, University of Potsdam
August-Bebel-Str. 88, 14482 Potsdam, Germany
johannes.wust@hpi.uni-potsdam.de

ABSTRACT

The fast-paced business environment in most industries requires analytical applications to report on the latest data of an enterprise. Today, organizations either rely on data warehouses, which make flexible reporting on up-to-date data almost impossible, or maintain copies of operational data, leading to increased costs and architectural complexity. In-memory database management systems (DBMS) on multicore CPUs are a promising platform to run analytical queries (OLAP) directly on operational data, in addition to processing load from transactional queries (OLTP). To meet response time requirements of both types of queries on a single system, some form of workload management has to be implemented to avoid system overload. Consequently, the early-stage PhD project described in this paper targets to design a workload manager optimized for mixed workloads on in-memory databases. At this stage, we envision a DBMS-internal workload management system that assigns as many resources to OLTP queries as necessary to meet response time requirements and schedules OLAP queries on the remaining resources. To minimize overall OLAP response time, we intend to leverage the underlying multicore architecture by scheduling available resources on the level of atomic database tasks, thereby considering cache behavior and the degree of parallelism. We believe that existing solutions in this area fall short for two reasons: (1) they do not consider the different response time requirements of both, operational and analytical applications, and (2) they do not take characteristics of modern in-memory DBMS, such as cache access patterns and massive parallel execution, into account. Our findings will be verified in the context of enterprise computing by using workloads from real-life enterprise systems.

1. INTRODUCTION

Traditionally, companies maintain their data warehouses separately from their operational databases. This is mainly due to different functional and performance requirements of both domains [6]. As businesses require for more flexible, ad-hoc reporting on up-to-date data, an ongoing trend towards analytical applications reporting on operational data, often referred to as operational business intelligence, becomes evident [11, 12, 18, 28, 34]. To avoid additional load on the operational database, these applications typically rely on copies of the operational data, so-called operational data stores [34]. Maintaining such a real-time copy is complex and expensive.

In-memory databases that leverage column-oriented storage provide the necessary performance to run analytical queries directly on the transactional database [27]. This enables building analytical capabilities on top of the transactional system, leading to reduced system complexity and reduced overall cost. Although queries are typically executed very fast on in-memory databases, execution time is still bound by bottleneck resources, such as main memory access, and depends on heavily on data locality [22]. Therefore, running a mixed workload of short-running, transactional queries and long-running, read-intensive analytical queries can lead to resource contention in the DBMS. To guarantee a flawless execution on an in-memory DBMS and save additional costs for an operational data store, some form of workload management is required.

Workload management for DBMS is a often discussed problem many database administrators struggle with. It becomes especially difficult if the DBMS faces heterogeneous queries. Common fields of research are workload management in the context of web requests [3, 24, 30] and business intelligence applications [4, 8, 18]. Queries originating from web requests have higher response time requirements as they directly add to the overall processing time of online requests. Within business intelligence applications, the main challenge is that query run-time varies from milliseconds up to a couple of hours [16].

We believe that the proposed solutions fall short for managing a mixed workload, consisting of transactional and analytical queries, on an in-memory database (see Section 2 on related work). As high transaction throughput is the key requirement for operational databases [6], we need to make sure that transactional queries are executed as fast as possible — therefore, controlling the execution of OLTP-style queries based on cost models or multiprogramming levels creates too much overhead for transactions that typically run only a few milliseconds. Consequently, we plan to directly control resource allocation on DBMS level and dynamically assign as many resources to OLTP queries required to guarantee expected throughput.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. This article was presented at: VLDB 2011 PhD Workshop, Aug 29 - Sep 3, Seattle, USA.

Copyright 2011.
OLAP queries have to be scheduled on the remaining resources as efficiently as possible. Here, existing work on admission control [8, 18, 29, 30] is an effective way to avoid overload and to notify users or database administrators if queries are deferred or rejected. Thus, we see this work as complementary to ours and assume a perfect admission control. However, an external admission control has two shortcomings: (1) it cannot adapt quickly on internal reallocation of resources between OLAP and OLTP, and (2) it cannot directly influence two key characteristics of efficient query execution in in-memory databases on multicore CPUs: cache sensitivity and parallelism. If not taken into consideration, parallel execution of two database operators can even be slower than sequential execution on a multicore CPU if they mutually evict cache lines [7, 21, 22]. Therefore, we propose an DBMS internal OLAP scheduler on the level of atomic database tasks which considers aspects such as cache sensitivity or CPU vs. memory boundness in its scheduling decision.

As an outline, the early-stage PhD project proposed in this paper targets to manage mixed workloads consisting of both, transactional and analytical queries, on in-memory DBMS. The objective is to make the following contributions:

- An analysis of the bottleneck resource in an overloaded in-memory database under varying workloads
- A design of a workload management system for in-memory databases under a mixed OLTP/OLAP workload with guaranteed OLTP response time
- An optimized scheduler for OLAP-style queries based on atomic database tasks, that works on dynamically changing number of cores, thereby considering parallel execution of tasks and overall cache access patterns
- A validation of results using real workloads
- An extension of the approach to a shared-nothing environment with distributed processing nodes

This paper is structured as follows: the reminder of this section briefly introduces in-memory databases and motivates analytical application on transactional data. Section 2 gives an overview of related work and Section 3 provides an overview of the envisioned workload management system. Section 4 outlines the research agenda, and Section 5 finishes with concluding remarks.

1.1 In-memory databases

We consider an in-memory DBMS as a database system where the primary persistence resides entirely in the main memory. In-memory databases are applied to specific problems with strong performance requirements for decades — e.g., they have been applied to applications such as routing tables for telecom providers, radar tracking or securities trading since the early 90s [10]. However, limited capacity of main memory has long prevented the storage of large data sets in memory and only data that is accessed frequently, so called hot data, was kept in memory. In recent years, the introduction of a 64 bit address space in commodity operating systems and the constant drop in hardware prices make large capacities of main memory in the order of terabytes technically feasible and economically viable. Together with ever increasing computing power due to multicore CPUs, this change enables the storage and processing of large sets of data in memory and opens the way for general-purpose in-memory data management.

A variety of commercially available and research databases have been developed. From a research point of view, MonetDB [23] and H-Store [15] have been the most influential systems; from a commercial perspective, SAP’s In-memory computing engine, IBM’s SolidDB and Oracle’s Times Ten are best known. Throughout the presented PhD project, we plan to extend the main memory storage engine HYRISE to validate our findings [13]. The storage engine follows a no-overwrite storage model and is prepared for query parallelism by implementing thread-safe data structures.

1.2 Analytics on operational data

An example for an enterprise application that greatly benefits from answering analytical queries from the transactional database is the availability check for products. Here, real-time stock levels from the transactional system are aggregated using OLAP queries to answer the question whether a specific customer order can be processed until a given due date. Further examples for applications producing a mixed workload in the context of enterprise computing are described by Krueger et al. in [17].

More general, analytics on operational data, can create opportunities in a wide range of other areas, such as gaining real-time insight into daily operations (revenues, margin, labor expenses), or identifying deviations on production processes in real-time [28].

2. RELATED WORK

Related work to the proposed project originates from three directions: (1) scheduling in real-time databases, (2) workload management in conventional databases, being the class of databases without specific real-time characteristics, and (3) cache conscious query processing. From an application point of view, our work is in the area of conventional database systems while building on top of existing work on cache conscious query processing. We consider basic characteristics of real-time systems as sufficiently different to distinguish our work: in real-time databases, transactions and data typically have time semantics, meaning they are valid only for a defined interval of time [32]. Notice that this is fundamentally different from time-requirements in conventional databases. Consider a banking application: even if a bank guarantees to a customer that a withdrawal from an ATM is immediately booked on the corresponding account, the transaction does not become invalid if the underlying DBMS cannot process in time.

This section first briefly summarizes work in the field of scheduling jobs in real-time databases, then discusses related work in the area of conventional databases, and finally relates our work to research in the field of cache conscious query processing.

2.1 Real-time databases scheduling

Much research has been conducted in the context of real-time databases to guarantee that time constraints are met. Stankovic et al. separate scheduling algorithms in two categories: static and dynamic, whereas static algorithms have complete knowledge about the set of tasks and the corresponding constraints [31]; dynamic algorithms consider changing set of tasks and their timing constraints. The most
prominent scheduling algorithm in the class of dynamic algorithms is Earliest Deadline First [20]; however, this algorithm performs best in moderately-loaded systems [14] and creates fairness problems for long running queries, especially if the query run-time is not available [35]. Advancements of this algorithm, Adaptive Earliest Deadline [14] and Adaptive Earliest Virtual Deadline [26] have been developed to address these shortcomings.

2.2 Conventional database workload management

Workload management in conventional databases can be divided into two main classes: external and internal. The general idea of external workload management is to control the number of queries that access the database (admission control). Internal workload management systems typically control the available resources, such as CPU or main memory, and assign them to queries.

Early work on internal workload management is published by Carey et al. [4, 5]. The simulation studies are specific to disk-based DBMS, as they extensively model disk-based DBMS characteristics such as disk rotation time or buffer management. A more recent work by McWherter et al. [24] shows the effectiveness of scheduling bottleneck resources using priority-based algorithms in a disk-based DBMS.

Recent work in DBMS workload management has mainly focused around admission control. Schroeder et al. [29, 30] propose an external queue management system that schedules queries based on defined service-levels per query-class and a number of allowed queries in the database, the so-called multiprogramming level. We share the basic idea with the concept of Niu et al. [25] that manages a mixed workload of OLTP and OLAP queries by controlling to OLAP queries depending on the response times of OLTP queries. Although external workload management systems are applicable to in-memory databases, we believe that these solutions fall short in our scenario, as they do not exploit information about resource utilization within the database. Long running queries can easily overload the database and a system entirely based on admission control has no possibilities to react if high priority queries, in our case OLTP queries, need to access the database.

The work of Dayal, Kuno and Krompass et al. [8, 9, 16, 18] focuses on the management of analytical queries in the area of data warehouses. It can be considered as a mixture of external and internal workload management. Besides admission control based on external query run-time prediction, they additionally control the execution of queries and propose the possibility to kill long running queries in case high priority queries are present. Focusing on data warehouses, the overhead from query run-time prediction and admission control is acceptable and we consider this as complementary to our approach for scheduling OLAP queries. However, we believe that the overhead imposed on short running, transactional queries is too high to achieve satisfying query response times. In their discussion around operational BI [8, 18], the underlying database is not the operational database of the organization, but rather a copy of this data store.

2.3 Cache conscious query processing

Ailamaki [2] and Manegold et al. have demonstrated the importance of data access patterns for in-memory computing [22]. Manegold et al. [21] proposed a generic cost model for cache access patterns, which is used in our work to identify atomic database tasks. Cieslewicz et al. analyzed the cache behavior of concurrently executed database operators with state, such as joins or aggregates, under a time-sliced scheduling [7]. Lee et al. [19] propose a disk-based database prototype on static hardware resources that minimizes shared cache conflicts. Both results make us confident that we can significantly reduce OLAP query execution time with our cache sensitive OLAP scheduling.

3. A BLUEPRINT FOR A WORKLOAD MANAGEMENT SYSTEM

This section describes a high-level blueprint for the workload management system of our in-memory database, designed to provide guaranteed OLTP response times, while efficiently scheduling OLAP queries. We first present a system overview and then describe our initial ideas for controlling OLTP response times and scheduling OLAP queries.

3.1 Overview

We consider a shared-nothing architecture of multiple computing nodes, each consisting of multiple multicore processors sharing a common main memory. We start introducing the concepts of workload management on the level of a single computing node. Extending the single-site concept to a distributed scenario is part of the research agenda (See section 4) Figure 1 illustrates the main components of our workload management system. Notice that outside of the database, a complementary external admission control can defer or reject OLAP queries in case they cannot be handled by the available resources efficiently — this is complimentary to our work and described in [8, 18, 29, 30]. Consequently, we assume that the database system is sufficiently sized to handle the peak OLTP load, and that a perfect admission control only admits as many OLAP queries as can be handled by the system. Inside the DBMS, we have two separate queues - one for transactional queries and one for analytical queries. They get filled by the output of the query compiler,
which translates OLTP queries into plan and OLAP queries into atomic database tasks, as discussed later in this section. The resource allocator assigns processing cores to each queue manager (abbreviated as QM in Figure 1), depending on the number of processing cores provided by system information of the underlying operating system. Each queue manager maintains a number of worker threads (usually as many as assigned processing cores), assigns tasks to the threads and pins them to cores. The execution controller collects information about query execution time. In case performance falls below or well above a defined threshold for an acceptable state, e.g., average query response time or OLTP throughput, the execution controller informs the resource allocator, which reallocates resources between the OLTP and OLAP queue manager. Although not further discussed in this paper, a component for concurrency control and one for recovery are shown to ensure consistency and fault-tolerance of the DBMS. The next sections describe the envisioned execution model for OLTP and OLAP queries in more detail.

3.2 OLTP queries

OLTP queries can run within milliseconds on in-memory databases. Under certain assumptions, in-memory databases can execute the entire OLTP workload in a single-threaded fashion, as discussed in [33] and demonstrated with H-Store [15]. Hence, we believe that predicting execution costs for queries imposes too much overhead on transactional queries. Consequently, we intend to allocate processing cores exclusively for OLTP queries and assign queries to threads according to a FIFO rule. The execution controller monitors query execution time and signals the resource allocator if queries run above or well below the threshold for acceptable execution time.

3.3 OLAP queries

OLAP queries have to be scheduled on the resources not used for transactional queries. These can dynamically change depending on the OLTP workload. Consequently, our units of scheduling need to be chosen rather fine-granular to allow a quick reaction on this changing environment, ideally without preempting tasks. Therefore, we plan to decompose a query into atomic database tasks, thereby considering key characteristics of in-memory databases and multicore CPUs; we call these tasks atomic as they have to run sequentially. We first introduce the concept of atomic database tasks, next define the parameters for our scheduling problem to solve and finally outline the envisioned scheduling component.

3.3.1 Atomic database tasks

Queries entering a DBMS are typically compiled to logical query plans consisting of relational database operators. This logical query plan is then converted into a physical query plan of operations, basically by choosing an algorithm for each operator and deciding on the execution order. Operations are not the right level for scheduling as they are typically further split for parallel execution, and different phases of an operation potentially show different cache access patterns [21].

Considering parallel execution is particularly important for OLAP queries which are typically long-running, read-intensive queries and operate on large datasets — especially if stored in a column-oriented style, parallelism can greatly reduce execution time [27].

Considering cache access patterns is especially important in the context of in-memory computing as demonstrated in [22, 21]. If we run tasks concurrently, we have to take care of their footprint size in caches; this is the number of cache lines it revisits — if two tasks run concurrently on a CPU with a combined footprint size larger than a shared cache, they mutually evict cache lines, resulting in decreased performance [7, 21, 22].

To account for both aspects in our scheduling decision, we need to partition a database operation into atomic tasks by the following criteria:

- The task shows one of the basic cache access patterns as defined in [21]
- The task cannot be further partitioned, or performance cannot be increased by further partitioning and parallel execution

The atomic database tasks need to be determined for each database operator. Therefore, we first split the sequential execution of operator implementation into tasks that show a varying cache access pattern. This split is fixed for each implementation and can be predefined at implementation time. Next, we determine the maximum degree of parallelism for each of these tasks; besides operator implementation, this degree depends on run-time parameters, such as the number of tuples of operands or the number of available OLAP cores; therefore, the query compiler needs to know the actual number of assigned cores to the OLAP QM to create the atomic tasks as shown in Figure 1. Notice that the upper bound is generally the number of cores, as further partitioning would not increase the run-time.

As an example, Figure 2 shows a join operation, implemented as a hash join, with a degree of parallelism of 3. The hash join is first split into two tasks with varying cash access patterns: \( \text{hash\_build} \), for building a hash table on the smaller relation \( U \) and \( \text{hash\_probe} \), for finding matching tuples larger table \( V \). Then each of these tasks is split further into atomic tasks, based on the maximum degree of parallelism — in this case 3; additionally a merge task has to be added to produce the final result table \( W \). As outlined in Section 4, splitting of database operators to atomic database tasks is part of the research agenda of this PhD project.

3.3.2 Parameters of the scheduling component

\[ \text{Notice that some tasks, such as building a hash table, read and write data and therefore can lead to more than one basic access patterns} \]
The goal of this section is to outline the important parameters of the OLAP scheduler. We define a machine environment, describing the resources and the task characteristics, describing the attributes of the tasks to be scheduled. The system resources considered in the scheduling decisions are the number of CPUs and the number of cores per CPU. The number of cores allocated to the OLAP scheduler can vary over time, depending on the OLTP workload. Considering up to 1 TB of RAM in commercially available servers, we assume the machine to be equipped with sufficient main memory for the working sets of each query. Notice that we do not model the bottleneck resource of memory-bound queries, memory bandwidth, explicitly; this follows the idea that processes can only issue requests on other resources after they have gained the CPU as illustrated in [1]. We also do not consider locks on database tables, as OLAP queries are typically read-mostly; due to the no-overwrite characteristic of the in-memory database, only queries that write data would require locks. An atomic database task as defined in Section 3.3.1 has the following characteristics:

- **Data dependency**: an atomic task is potentially dependent on other tasks; we differentiate between a strict sequential ordering and the potential for pipelining
- **Cache footprint size**: the number of cache lines the task potentially revisits as defined in [21]
- **Bottleneck resource**: the resource, which limits the execution time of the task (CPU or memory bandwidth)
- **Core affinity**: a preference for a specific core — this is typically the core that is closest to the data in a NUMA environment

### 3.3.3 Scheduling vision

The scheduling problem is to allocate the tasks in the OLAP queue to the resources assigned to the OLAP queue manager by minimizing overall execution time. Potential triggers for the scheduler are if a new query enters the database or if the available resource for OLAP queries change. The scheduler’s task is to make a nearly-optimal scheduling decision in the shortest possible time frame. At this stage of the project, we can define a number of desired properties of a schedule, leaving the design of the scheduling algorithm an open research problem for this PhD project:

- The number of tasks assigned to a CPU should not have a combined cash footprint size greater than the shared cache
- The number of tasks assigned to a CPU should be balanced with respect to their bottleneck resource
- Pipelining of data dependent tasks should be applied when possible

### 4. RESEARCH AGENDA

This section details the main open research questions addressed in this PhD project based on the expected contributions introduced in Section 1.

**Bottleneck analysis for in-memory databases.** It is crucial to understand the bottlenecks in a system to manage its resources efficiently. Therefore, the first step in this project is to measure the bottleneck for our in-memory database under various workloads. We plan to measure the time queries are waiting for resources, such as CPU, memory-bandwidth and locks under a mixed workload. We plan to generate a mixed workload of OLTP and OLAP queries from industry benchmarks (TPC-C and TPC-H), as well as to test real-life workloads from transactional systems and business warehouses from large companies.

**Design of a workload management system with guaranteed response times for OLTP queries.** In the next step, we plan to implement the required infrastructure, such as thread pools, performance control of queries, query queues as introduced in Figure 3. The focus will be on running the OLTP queries with minimized overhead, potentially using a single thread for queries that change data to minimize the need for concurrency control.

**Optimize OLAP scheduling for multicore CPUs.** The next phase is about leveraging the characteristics of multicore CPUs for OLAP scheduling, as described in section 3. Therefore, we study the scale-up behavior and cache access-patterns to determine the split of a number of common query operators like scans, aggregations, and a join implementation into atomic database tasks. Based on these, we demonstrate the expected performance increase compared to simple scheduling approaches. The next step is to design a cost model that allows us to predict the execution cost of a schedule and can be used to determine nearly-optimal schedules.

**Validation of results.** We plan to validate the feasibility of our proposed approach by using real-life data of a large company. The objective is to fill the in-memory database with transactional data of a specific domain, such as sales items, and run the transactional workload as recorded on the companies productive transactional system. Additionally, we take the analytical workload that runs against the corresponding data in the data warehouse and execute it on the in-memory database holding the transactional data. As the transactional database does not consist of aggregated cubes typically present in a warehouse, we have to translate the analytical queries to queries that run against the transactional data but produce the same result.

**Extension to distributed shared-nothing architecture.** As a final step, we plan to extend the single-site concept to a distributed environment of processing nodes that do not share any resources. We believe that the concept presented here can be equally applied in a distributed environment. However, we have to consider two issues:

- The distribution of queries to processing nodes depends on data partitioning over processing nodes
- The process that assigns tasks to processing nodes can easily become a single point of failure

### 5. CONCLUSION

As outlined in this paper, the planned PhD project intends to contribute a workload management system for in-
memory databases to allow the processing of transactional and analytical queries on the same database. We consider this an important contribution towards a wide-ranging use of in-memory databases within enterprise computing, leading to more flexible and up-to-date reporting on operational data, as well as reduced IT costs, as copies of operational data for reporting become obsolete.

6. REFERENCES


