

Prof. Hasso Plattner



A Course in In-Memory Data Management

The Inner Mechanics
of In-Memory Databases

September 4, 2015

This learning material is part of the reading material for Prof. Plattner's online lecture "In-Memory Data Management" taking place at www.openHPI.de. If you have any questions or remarks regarding the online lecture or the reading material, please give us a note at openhpi-imdb@hpi.uni-potsdam.de. We are glad to further improve the material.

Research Group "Enterprise Platform and Integration Concepts",
<http://epic.hpi.uni-potsdam.de>

Workload Management and Scheduling

22.1 The Importance of Speed

One of the most important factors that determine the usability for interactive applications is response time. Psychological studies show that the acceptable maximum application response time for a human is about three seconds. After three seconds, the user might lose concentration and do something else. Once the application has finished its processing and is ready for the next human input, the user has to refocus on the application again. These context switches are extremely expensive as they constantly interrupt the user and lead to being unproductive. Consequently, an important objective for data management is to answer queries issued by interactive applications fast.

As described in Chapter 3, it is commonly distinguished between analytical and transactional applications. Both types of applications issue a specific workload to the underlying database. Although not formally defined, it is widely agreed that transactional queries are considered as short-running, whereas analytical queries are typically being classified as long-running. Due to their specific fields of application, they also differ in their service level objectives: Transactional applications are often coupled to customer interactions or other time-critical business processes and have high response time requirements. On the other hand, analytical applications that generate reports are typically considered batch-oriented with comparably less critical response-time requirements. As introduced in Chapter 2, new requirements in enterprise computing demand for fast execution of complex, analytical queries. A table scan necessary for an OLAP query can easily be parallelized to a high degree by a query optimizer. If the query optimizer decides that the query can be executed in parallel, it separates the query into multiple sub-queries, executes every sub-query in parallel and combines their intermediate results, resulting in a reduced overall execution time. This parallel

execution is required to reach response times of merely seconds, as expected by interactive users.

Obviously, processing a mixed workload of response time critical transactions, as well as complex analytical queries can potentially lead to resource contention in the database system. Based on the query classes discussed above, we can state a number of desirable design objectives for a database system executing a mix of these query classes. First, queries need to run concurrently to efficiently use available resources: as transactional queries typically run sequentially, allowing only one query at a time would result in an underutilization of the system. Also, analytical queries or other database tasks will have parts that cannot be parallelized to a degree that leverages all available processing units. Second, transactional queries need to run with a higher priority over other query classes: given the relatively short run-time of transactional queries compared to the other classes, they cannot wait until queries of the other classes have completed to meet their service level objectives. Third, interactive analytical queries have to gain access to a large number of resources quickly: given the tight response time requirements and the high complexity, queries of this type need to be executed with a high degree of intra-query parallelism and therefore require access to a larger number of available processing units. And fourth, if a query is running on a number of processing units in parallel, it needs to free up resources quickly in case a query with higher priority arrives for execution. Consequently, some form of workload management is required in the database systems to enforce these requirements.

22.2 Scheduling

The database scheduler is the component that decides which of the incoming query or sub query gets executed on the available computing resources. In general, scheduling goes along with several kinds of complications: interdependencies between queries, different resource utilization for the queries (e.g. some are CPU-bound or memory-bandwidth bound), restricted resources, locality of operations (e.g. a filter task should be executed at the node where the data is stored) and so on. These possible complications make finding an optimal execution plan a very complex task which is NP-hard in most cases. The problem of finding an optimal scheduling becomes even more complex, if we consider the online arrival of multiple queries. However, approximate solutions often outperform complicated algorithms in practice.

To achieve the desired system characteristics as introduced in the previous section, the scheduler follows a task based execution model. The logical query plan of an incoming query is transformed into a set of atomic tasks that represent this plan. These tasks may have data dependencies, but can

be executed otherwise independently, thus forming a directed acyclic graph. A task-based approach provides two main advantages over considering a whole query as the unit for scheduling: More fine granular units of scheduling allow for better load balancing on a multiprocessor system, as well as more control over progress of query execution based on priorities. Splitting queries into small units of work introduces points in time during query execution, where lower priority queries can be paused to run higher priority queries without the need of canceling or preempting the low priority query. A more detailed discussion of task based scheduling can be found in [WGP13].

22.3 Mixed Workload Management

Typically, optimizing resource usage and scheduling is a workload-dependent problem and becomes even more complicated when two different types of workloads are mixed. As said before, a transactional workload is characterized by short running queries that must be executed within tight time constraints. In contrast, an analytical workload consists of more complex and computationally heavier queries. Running mixed workloads on a single database instance leads to potentially conflicting optimization goals. The response time for transactional queries must be guaranteed. This can be achieved by assigning priorities to tasks corresponding to critical transactional queries and adopt a priority scheduling policy in the task scheduler. At the same time, the response time for analytical queries should be as short as possible. This is achieved by using the aforementioned task based scheduler, which makes sure that the available resources are fully utilized.

22.4 References

- [WGP13] Johannes Wust, Martin Grund, and Hasso Plattner. Tamex: a task-based query execution framework for mixed enterprise workloads on in-memory databases. In *GI-Jahrestagung*, pages 487–501, 2013.

