HyDash: A Dashboard for Real-Time Business Intelligence based on the HyPer Main Memory Database System

Maren Steinkamp Tobias Mühlbauer*

Lehrstuhl für Datenbanksysteme
Technische Universität München · Boltzmannstraße 3 · D-85748 Garching
{steinkam, muehlbau}@in.tum.de

Abstract: Business Intelligence (BI) is a set of techniques that help improve business decision making. From a technical point of view, BI relies on a set of tools which includes performance dashboards: layered services that combine monitoring, analysis, and reporting. However, most dashboard solutions today are based on data warehouses which face a problem of data staleness—a circumstance caused by the separation of transactional and analytical processing. Novel hybrid main memory database systems such as SAP's HANA or HyPer overcome this issue and achieve best-of-breed transaction processing throughput and analytical query response times in one system in parallel on the same database state. In this work, we contribute HyDash, a web-based dashboard for real-time business intelligence. HyDash visualizes key performance indicators (KPIs) based on predefined SQL queries and provides an ad-hoc SQL query interface to dynamically extend the scope of the dashboard. HyDash relies on (i) hybrid main memory database systems and (ii) a push-based query result update mechanism based on the novel HTML5 WebSockets.

1 Motivation

Business Intelligence (BI) systems are data-driven decision support systems, i.e., concepts and methods that improve business decision making by using fact-based support systems [Pow03], which, e.g., analyze an enterprise's transactional data by means of data analysis. Eckerson [Eck10] names an overview of BI tools: reports, performance dashboards, online analytical processing (OLAP), ad-hoc querying, visual analysis, search, as well as statistical models. In his view, performance dashboards represent "the latest incarnation of BI" and provide "a layered information service that combines monitoring, analysis, and reporting" whereby meeting most of the information needs of casual BI users. Commercial dashboard solutions are developed by established companies such as SAP Business Objects or IBM DB2 Alphablox. The advent of big data and an increased need for complex analysis and data visualization recently triggered the launch of several startups such as Platfora (`http://www.platfora.com/`) or Chartio (`http://www.chartio.com`) who now also develop BI systems and dashboard solutions. These dashboards display key performance indicators (KPIs) to analysts in a way that is easy to interpret [PCMP07].

*Tobias Mühlbauer is a recipient of the Google Europe Fellowship in Structured Data Analysis, and this research is supported in part by this Google Fellowship. This work has further been partially sponsored by the German Federal Ministry of Education and Research (BMBF) grant HDBC 01IS12026.

However, the selection of these KPIs is often static and the KPIs are, as most dashboards are based on an underlying data warehouse solution, rarely updated. This issue is often referred to as the problem of *data staleness* which is due to the common separation of transactional database systems and data warehouses. Long-running analytical queries (OLAP), e.g., those determining the KPIs, are running in the data warehouse so to not interfere with the mission-critical online transactional processing (OLTP) in the transactional database. The data warehouse is thereby periodically updated with a fresh transactional view by a so-called extract-transform-load (ETL) phase. As the ETL phase itself could again interfere with OLTP processing, it is often only carried out during times of reduced OLTP load. During busy hours, the data warehouse is not updated and is thus inevitably facing the problem of data staleness. Industry leaders such as Hasso Plattner of SAP however argue that the issue of data staleness is inappropriate for *real-time business analytics*[1] [PZ11]. Recent developments in main memory database systems address this problem: hybrid OLTP&OLAP database systems such as SAP's HANA or HyPer [KN11] achieve best-of-breed transaction processing throughput and OLAP query response times in one system in parallel on the same database state. Being able to access KPIs based on the newest available transactional data generates an enormous benefit for decision makers in industry and science.

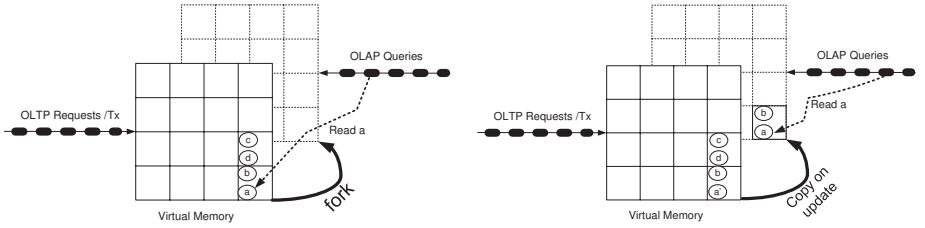
In this work we contribute HyDash, a web-based dashboard for real-time business intelligence. It is the goal of HyDash to visualize key performance indicators (KPIs) based on predefined SQL queries and to further provide an ad-hoc SQL query interface to dynamically extend the scope of the dashboard. Furthermore, updates in query results triggered by changes in transactional data should be propagated to the dashboard in real-time. To achieve this goal, HyDash relies on (i) hybrid main memory database systems, which efficiently separate query processing from mission-critical transactional processing and provide low response times and (ii) a push-based query result update mechanism based on the novel HTML5 WebSockets [Hic12b], which provide a full-duplex communications channel between a web application and its backend. In order to achieve a performant web frontend, several technologies were combined. On the backend side, the HyDash implementation is based on the hybrid main memory database system HyPer and its scale-out ScyPer [MRR+13]. However, any other hybrid main memory database system could have been used. Finally, we evaluate the dashboard by conducting a case study based on the CH-BenCHmark [C+11], a combination of the TPC-C and TPC-H benchmarks.

The remainder of this paper is structured as follows: Section 2 introduces the HyPer hybrid OLTP&OLAP main memory database system. Section 3 outlines the architecture and evaluation of HyDash. Section 4 then provides an overview of work related to HyDash. We conclude by summarizing our results and sketching further work.

2 The HyPer Hybrid OLTP&OLAP Main Memory Database System

HyPer [KN11] is a hybrid OLTP&OLAP main memory database system that achieves best-of-breed transaction processing throughput and OLAP query response times in one

[1] We use real-time in the sense of human real-time, i.e., response times that matter in the human decision making process.



(a) Forking a new snapshot (b) Copy on update to preserve consistency

Figure 1: HyPer's snapshotting mechanism

system in parallel on the same database state. OLAP query processing is separated from the mission-critical OLTP processing — for which the ACID properties are guaranteed — through a virtual memory (VM) snapshotting mechanism. On modern commodity hardware HyPer achieves a throughput exceeding 100,000 TPC-C transactions per second while providing query response times comparable to market-leading OLAP databases.

HyPer's performance is due to the following key characteristics:

1. HyPer relies on in-memory data management and eliminates the ballast caused by DBMS-controlled page structures and buffer management in traditional database systems.

2. OLAP processing is separated from mission-critical OLTP processing by forking transaction (TX)-consistent VM snapshots using the POSIX system call `fork()` (Figure 1(a)). The hardware- and operating system-supported "copy on update" mechanism then preserves the consistency of the snapshot by copying pages only if a page is modified (Figure 1(b)). The creation of each snapshot is scheduled as a system-internal transaction and as such no concurrency control mechanism is needed to separate OLAP and OLTP processing.

3. Transactions and queries are specified in SQL or, alternatively, in the HyPer scripting language which treats SQL as a first class citizen. HyPer essentially acts as a compiler that efficiently compiles these transactions and queries into optimized machine-independent LLVM assembler code [Neu11].

4. Parallel transactions are separated via lock-free admission control. Non-conflicting transactions can thus be processed simultaneously in multiple threads without the need for concurrency control.

5. HyPer uses logical redo logging where the invocation parameters of transaction procedures are logged via a high-speed network.

3 HyDash: A Dashboard for Real-Time Business Intelligence

HyDash is a web-based dashboard for real-time business intelligence which visualizes key performance indicators (KPIs) based on predefined SQL queries and provides an ad-hoc SQL query interface to dynamically extend the scope of the dashboard. Updates in

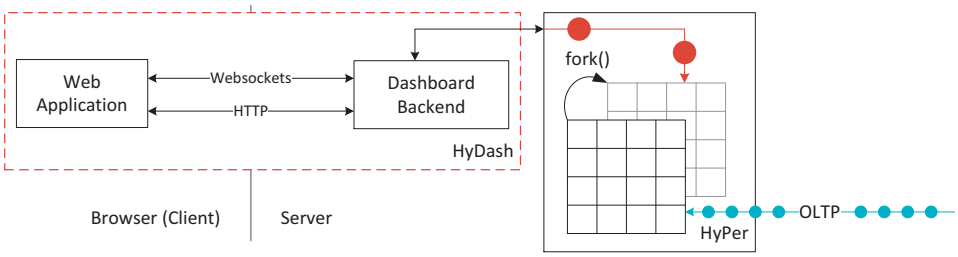


Figure 2: HyDash architecture

query results triggered by changes in transactional data are propagated to the dashboard in real-time through HTML5 WebSockets [Hic12b]. On the backend side, the HyDash implementation is based on the hybrid main memory database system HyPer and its scale-out ScyPer [MRR+13]. However, any other hybrid main memory database system could have been used.

In the following, we first present HyDash's client-server architecture which consists of a web application and a backend application that is directly communicating with the underlying hybrid main memory database system. Then, we evaluate HyDash by conducting a case study based on the CH-BenCHmark [C+11], a combination of the TPC-C and TPC-H benchmarks. We thereby use the 22 analytical TPC-CH queries of the benchmark as the predefined KPIs of the dashboard.

3.1 Architecture

HyDash is implemented using a client-server architecture approach. It consists of a web application and a server-side dashboard backend component. The backend component thereby connects the HyPer main memory database system with the web application. The web application and dashboard backend component communicate through HTML5 WebSockets [Hic12b] and HTTP requests. HTML5 WebSockets provide a low-latency and low-cost full-duplex communication channel between the web application and the dashboard backend component. We use HTML5 WebSockets to implement a scalable push-based approach to propagate changes in query results triggered by updates in transactional data in real-time. HTTP is solely used to deliver the web application's initial page which contains the JavaScript application code to communicate through WebSockets. All other communication related to the visualization of KPIs and ad-hoc query results uses WebSockets. The architecture of HyDash is shown in Figure 2.

Web application. The HyDash web application is designed using the HTML5 markup language to structure the document, CSS3 for the design, and JavaScript to define the application behavior. For each predefined KPI, the dashboard provides a separate view. Each of these views visualizes the current query results in a columnwise sortable tabular view and additionally shows the last execution time of the query as well as statistics such as the amount of returned tuples. If applicable, the result is further visualized as a chart. For the charts and tabular view, we use the Google Chart Tools (`https://developers.`

google.com/chart/). Charts can, e.g., be pie, line, bar, or world map charts. For predefined KPIs, the types of charts that can be used are statically defined. In the future we also plan to implement a chart recommendation engine that proposes suitable chart types based on the returned result set. Additionally, the web application provides an ad-hoc query interface where users can enter arbitrary SQL queries. These queries then define a new KPI and the dashboard visualizes the query results and its updates in real time. Currently only a tabular view is used to display the result. With the aforementioned chart recommendation engine, we plan integrate charts in the ad-hoc query interface.

The web application is communicating with the dashboard backend through HTML5 WebSockets. This allows a push-based communication where the dashboard backend pushes updated query results to connected web applications. The push interval for each query is based on the query's average execution time. On the web application side, we use the Google Closure (https://developers.google.com/closure/) implementation of WebSockets; on the dashboard backend side we use a C++ implementation of the WebSockets protocol. Whenever the user switches the page to see another query's result, the web application checks whether it already cached the query results for that query and loads the data table. Additionally, it requests the most recent query result from the backend and therewith subscribes for updates for this specific query. As soon as an update arrives on the WebSocket, the JSON serialization is parsed and the tabular view's and charts' data are updated automatically. Additionally, in order to reduce the tabular views' rendering time, the table is paged with a page size of 20. This clear arrangement also makes it easier for the user to browse the data. When new data for the table arrives, the table remains on the currently displayed page.

Dashboard backend. The web clients communicate with the dashboard backend through WebSockets and HTTP requests. The dashboard backend is the central instance that handles all open connections, coordinates the execution of the queries with the underlying HyPer main memory database system, and broadcasts updated query results to all connected web clients that subscribed for that query. The dashboard backend is implemented in C++ and is based on a lightweight webserver which provides access to the web application through HTTP. It also implements a WebSocket server using the WebSocket++ (http://www.zaphoyd.com/websocketpp) implementation which is used to broadcast updated query results. The communication between the dashboard backend and the database system is pull-based in order to allow portability to other database systems. In the following we describe the processes in the dashboard backend in more detail.

Preparation of the query execution. On startup, the dashboard backend prepares the execution of the predefined KPIs. Therefore it reads the queries for the KPIs from files and sends them to the HyPer database system. Once HyPer confirms that a query is prepared, the dashboard backend saves the identifier of the prepared query as well as the schema information for the query result as JSON strings. Once all confirmations have been received, the continuous execution of the predefined queries is started.

Executing the predefined queries. The dashboard backend requests the execution for each of the prepared queries. This step is continuously repeated after query results are received

	Mean time (standard deviation) [ms]			
Query	17	12	2	10
Result set size [tuples]	1	12	1425	58398
Parsing	0 (0)	0.3 (0.48)	21.5 (3.31)	256.6 (49.62)
Initial build of table	3.7 (1.95)	3 (1.41)	1.7 (0.67)	4.5 (2.17)
Initial build of chart	2.4 (1.96)	2.2 (2.25)	-	-
Table update	3.9 (1.37)	8.2 (3.55)	30.2 (8.89)	359.2 (60.59)
Chart update	14.8 (4.85)	36.8 (16.37)	-	-
Total	19.9 (5.93)	40.2 (17.16)	53 (11.19)	634.8 (110.88)

Table 1: Parsing and rendering times of selected queries in HyDash.

and a request for a fresh snapshot is sent. HyPer only creates a new snapshot if the timestamp of the latest write transaction is greater than the the latest snapshot time, i.e., the current snapshot is outdated.

Postprocessing of query results. Whenever a query result arrives, the query result is transformed into a JSON string that can be sent directly to all subscribed web clients. Hereby the previously stored schema information including the type of each field allows to save numeric data types as numbers in the JSON string. This step is necessary to enable the direct loading of data into the Google Data Table that serves as data basis for the displayed tabular view (Google Table) and Chart. The JSON string is cached together with the query identifier. This allows an instant response to requests for specific query results coming from new web clients.

Handling requests from web-clients and broadcasting new query results. The dashboard backend communicates with the web-clients via a WebSocket. New connections on the WebSocket are handled by worker threads. If a request for a predefined query is received, the cached query result is immediately forwarded to the client. If the request contains an ad-hoc query request, the query is sent to the database system and will from now on be executed continuously until the client disconnects, i.e., the new ad-hoc query is treated such as a predefined KPI and the aforementioned steps are also processed for the new KPI. In both cases, for predefined KPIs and ad-hoc queries, the backend stores a mapping between clients and the queries these clients are interested in. This mapping is used to multicast updated query results only to the clients that subscribed for the query.

3.2 Evaluation: The CH-BenCHmark Case Study

To evaluate the performance of HyDash, a case study based on the CH-BenCHmark, a combination of the TPC-C and TPC-H benchmarks, has been conducted. The 22 TPC-H queries are treated as predefined KPIs. A screenshot of the predefined KPI visualization is shown in Figure 3. The ad-hoc query interfaces can be seen in Figure 4.

We conducted an evaluation of the dashboard on an AMD Bulldozer (8 cores, 32 GB main memory) commodity machine. Thereby the HyPer main memory database system, the dashboard backend, and the web browser were running on the same machine. To simulate transactional load, 50,000 transactions per second were generated and sent to HyPer. The queries from the dashboard backend were repeatedly processed in parallel on

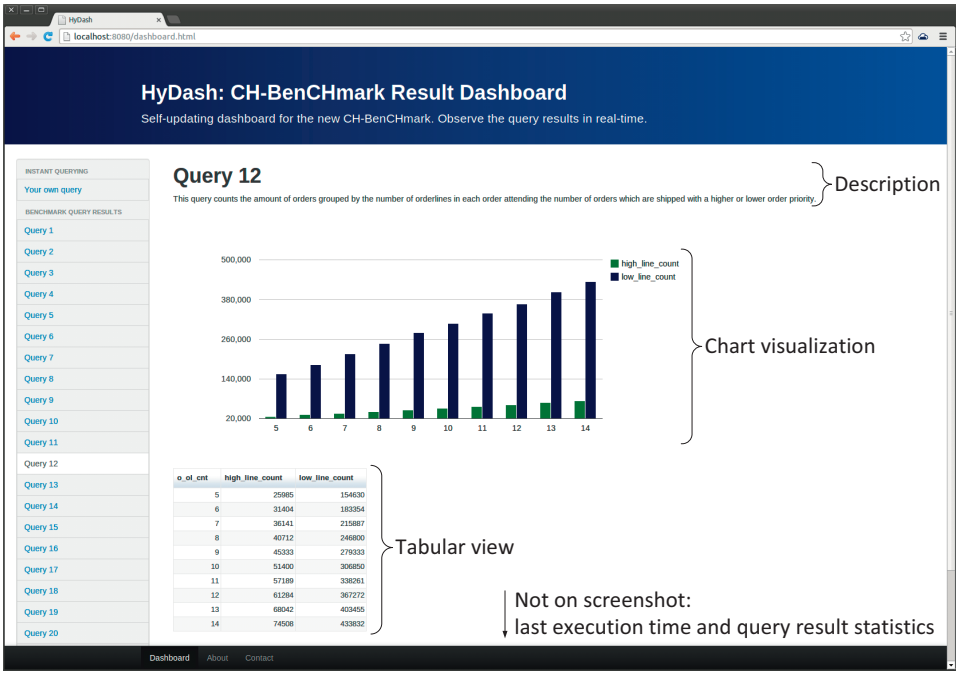


Figure 3: HyDash web frontend: predefined KPI visualization.

fresh snapshots. The majority of the query response times lay in the area of subseconds. We could not identify any performance issues related to the dashboard backend and the underlying HyPer database system. However, some performance bottlenecks could be revealed in the web application. The parsing of query results serialized as JSON strings is expensive and can, for large query results, take longer than the average query response time interval that produces the result. We experienced that total parsing and rendering times below 350 ms are not perceived by the user, whereas longer times, as, e.g., for Query 10 may freeze the webbrowser. One improvement is to parallelize the parsing using HTML5 Web Workers [Hic12a]. This allows the execution of parallel tasks in the web application without blocking the user interface. Displaying the query results in data tables or charts requires table and charting components that support a fluent data update without a high rendering time. The Google Table and Chart used in HyDash works well on data updates for a reasonable amount of data and provide good possibilities to show the transition of data in charts. Our evaluation showed that tables with up to 5,000 tuples allow subsecond update times, whereas bigger results can lead to increased rendering times that can exceed the average query response time that produces the result. The rendering time has been reduced by applying client-side paging for the data table and can be further optimized by enabling server-side paging of the query results. With these optimizations it is possible to efficiently visualize KPIs and query results in real-time. However, in cases where the parsing and rendering time exceeds the query's execution time, the time interval for the execution of the query must be adapted to avoid excessive demands towards the web application. Table 1 displays the parsing and rendering times of four selected queries.

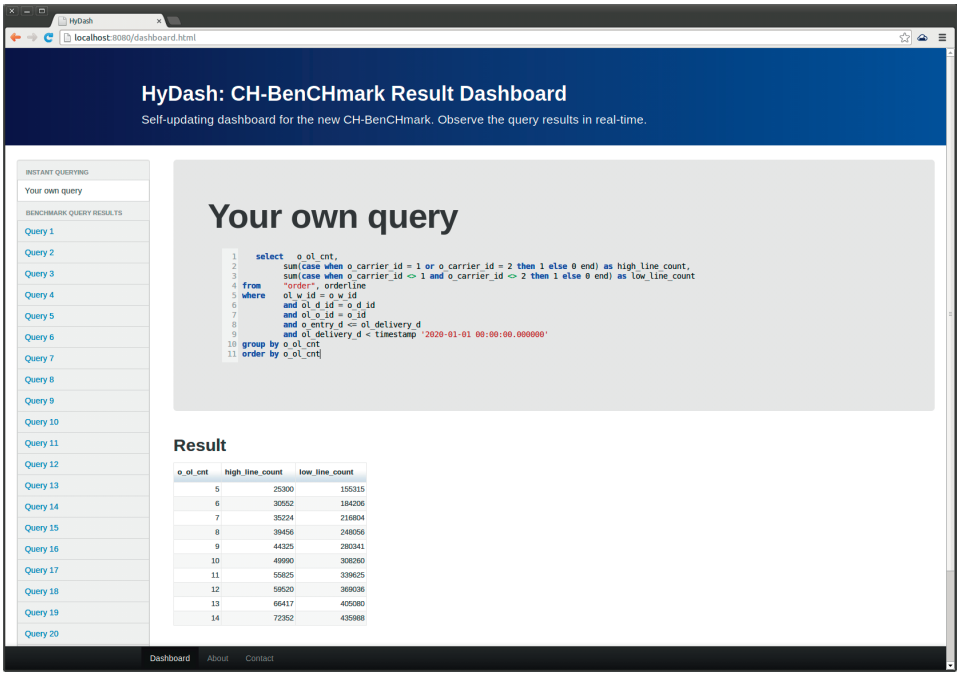


Figure 4: HyDash web frontend: ad-hoc query interface.

The evaluation is based on the non-optimized version of HyDash with only client-side paging enabled.

4 Related Work

BI is an umbrella term that, from a technical point of view, describes a variety of decision support tools [Eck10]. While, as described in Section 1, Eckerson believes that performance dashboards represent "the latest incarnation of BI", they often only provide an interface for various underlying technologies. HyDash is essentially a KPI visualization and ad-hoc OLAP query interface for hybrid main memory database systems which combine the functionality of a transactional database and a data warehouse. Besides relational data warehouses, BI tools can also be backed by, e.g., MapReduce-based systems [CLL+11, BEH+10], knowledge warehouses [NSIH02], or domain-specific applications.

Even though HyDash allows the user to dynamically extend the dashboard through ad-hoc queries, most KPIs are statically defined. Model-driven dashboard development approaches [PCMP07, PS07] aim at the automatic generation of dashboard implementations from business process models. In [PCMP07], Palpanas et al. introduce an approach to generate IBM Websphere-based dashboards from business process models. In contrast to model-generated dashboards, HyDash is a standalone solution that could however be integrated in a model-driven approach as a template.

In [CZ08], Chieu and Zeng identify the need for real-time data analysis and providing historical information for KPIs in performance monitoring dashboards. To achieve the goal

of real-time monitoring, they introduce a novel capture-transform-update (CTU) phase that replaces the ETL phase to update their data warehousing backend. HyDash is based on hybrid main memory database systems. These systems can efficiently provide real-time analysis capabilities. E.g., HyPer relies on relies on a virtual memory (VM) snapshotting mechanism that is efficiently supported by hardware and the operating system and does not face scalability issues that can occur when using CTU phases.

HyDash distinguishes itself from related web-based dashboards by providing human real-time analytics and push-based communication using HTML5 WebSockets. It can be dynamically extended by user-defined ad-hoc queries and can visualize complex OLAP query results with real-time update capabilities.

5 Outlook and Concluding Remarks

The current implementation of HyDash is only a starting point for a full-fledged real-time business intelligence (BI) dashboard. It can be extended in several directions. First, HyDash is currently based on the HyPer main memory database system, but can easily be integrated with other hybrid OLTP&OLAP database systems. Furthermore, there is huge potential to improve the dashboards's visualization capabilities, usability, and interactivity. Regarding data visualization, we plan to let not only users dynamically decide on the visualization format for a specific query result but rather also provide a recommendation system that, by using schema and query information, aids the user in choosing the right visualization format. HyDash's ad-hoc querying functionality provides a starting point to extend the dashboard with data exploration features by recommending new meaningful queries based on the user-defined queries as well as crowd-, statistical-, and OLTP as well as OLAP hotspot-knowledge. Especially the scientific community is asking for more efficient data exploration tools and methodologies [Abb09, KIML11]. Whereas the data acquisition problem is considered to be solved, meaningful data analysis and data exploration remains a big challenge. Data exploration aims at identifying relevant information within a huge amount of data [Abb09]. In [KIML11], Kersten et al. propose new approaches to handle data exploration more efficiently such as query morphing and one-minute kernels. We plan to extend the ad-hoc query interface of HyDash and the underlying HyPer main memory database system to implement these data exploration capabilities.

In this work we presented HyDash, a web-based dashboard for real-time business intelligence. HyDash visualizes key performance indicators (KPIs) based on predefined SQL queries and further provides an ad-hoc SQL query interface to dynamically extend the scope of the dashboard. To reflect transactional updates in KPIs and ad-hoc query results in real-time, HyDash relies on (i) hybrid main memory database systems, which efficiently separate query processing from mission-critical transactional processing and provide low response times and (ii) a push-based query result update mechanism based on the novel HTML5 WebSockets. On the backend side, the HyDash implementation presented in this work is based on the hybrid main memory database system HyPer — however, any other hybrid main memory database system could have been used. We evaluated the dashboard implementation by conducting a case study based on the CH-BenCHmark, a combination of the TPC-C and TPC-H benchmarks. The evaluation revealed that initially the visualization process in the web-based part of HyDash was not able to keep up with the rate query

result updates where propagated by the backend. This was due to the expensive parsing of large result sets and an increased rendering time of charting components for large result sets. For both issues solutions were proposed.

References

[Abb09] M. R. Abbott. A new path for science? In T. Hey, S. Tansley, and K. M. Tolle, editors, *The Fourth Paradigm*, pages 111–116. Microsoft Research, 2009.

[BEH+10] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke. Nephele/PACTs: a programming model and execution framework. In *SOCC*, pages 119–130, 2010.

[C+11] R. Cole et al. The mixed workload CH-benCHmark. In *DBTest*, pages 8:1–8:6, 2011.

[CLL+11] B. Chattopadhyay, L. Lin, W. Liu, S. Mittal, P. Aragonda, V. Lychagina, Y. Kwon, and M. Wong. Tenzing: A SQL Implementation On The MapReduce Framework. In *PVLDB*, pages 1318–1327, 2011.

[CZ08] T. C. Chieu and L. Zeng. Real-time performance monitoring for an enterprise information management system. In *ICEBE*, pages 429–434, 2008.

[Eck10] W.W. Eckerson. *Performance dashboards: measuring, monitoring, and managing your business*. Wiley, 2010.

[Hic12a] I. Hickson. Web Workers. `http://dev.w3.org/html5/workers/`, Editor's Draft November 25, 2012.

[Hic12b] I. Hickson. The WebSocket API. `http://dev.w3.org/html5/websockets/`, Editor's Draft November 24, 2012.

[KIML11] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou. The Researcher's Guide to the Data Deluge: Querying a Scientific Database in Just a Few Seconds. *PVLDB*, 4(12):1474–1477, 2011.

[KN11] A. Kemper and T. Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *ICDE*, pages 195–206, 2011.

[MRR+13] T. Mühlbauer, W. Rödiger, A. Reiser, A. Kemper, and T. Neumann. ScyPer: A Hybrid OLTP&OLAP Distributed Main Memory Database System for Scalable Real-Time Analytics. In *BTW*, 2013.

[Neu11] T. Neumann. Efficiently compiling efficient query plans for modern hardware. *PVLDB*, 4(9):539–550, 2011.

[NSIH02] H. R. Nemati, D. M. Steiger, L. S. Iyer, and R. T. Herschel. Knowledge warehouse: an architectural integration of knowledge management, decision support, artificial intelligence and data warehousing. *Decision Support Systems*, 33(2):143–161, 2002.

[PCMP07] T. Palpanas, P. Chowdhary, G. Mihaila, and F. Pinel. Integrated model-driven dashboard development. *Information Systems Frontiers*, 9(2):195–208, 2007.

[Pow03] D.J. Power. A Brief History of Decision Support Systems. `http://DSSResources.COM/history/dsshistory.html`, version 2.8, May 31, 2003.

[PS07] D.J. Power and R. Sharda. Model-driven decision support systems: Concepts and research directions. *Decision Support Systems*, 43(3):1044–1061, 2007.

[PZ11] H. Plattner and A. Zeier. *In-Memory Data Management: An Inflection Point for Enterprise Applications*. Springer, 2011.