

# Monopedia: Staying Single is Good Enough – The HyPer Way for Web Scale Applications

Maximilian E. Schüle\*   Pascal M. N. Schliski\*   Thomas Hutzelmann\*   Tobias Rosenberger\*  
Viktor Leis†   Dimitri Vorona†   Alfons Kemper†   Thomas Neumann†

\* Technical University of Munich, LMU Munich, University of Augsburg   † Technical University of Munich

\* {m.schuele,p.schliski,t.hutzelmann,tobias.rosenberger}@tum.de   † {leis,vorona,kemper,neumann}@in.tum.de

## ABSTRACT

In order to handle the database load for web scale applications, the conventional wisdom is that a cluster of database servers and a caching layer are essential. In this work, we argue that modern main memory database systems are often fast enough to consolidate this complex architecture into a single server (plus an additional fail over system). To demonstrate this claim, we design the *Monopedia Benchmark*, a benchmark for web scale applications modeled after Wikipedia. Using this benchmark, we show that it is indeed possible to run the database workload of one of the largest web sites in the world on a single database server.

## 1. INTRODUCTION

In terms of request rates, large web sites like Wikipedia are some of the most demanding use cases for database systems. Applications with very high request rates are indeed often called *web scale*. To handle the database load of web scale applications, the canonical approach is to use a scale-out architecture, i.e., to distribute the database across a number of servers [7]. This is often done in a master-slave architecture, in which writes are propagated from the master to the slaves. The slaves allow one to scale the number of read requests, which, for most web sites, are much more common than writes. Furthermore, since traditional, disk-based database systems in combination with PHP are often too slow for large web applications [4], an additional caching layer (e.g., consisting of in-memory caches like Memcached, or proxy servers like Squid) is used, reducing the load on the database system [6].

In this work, we argue that modern in-memory database systems are a viable alternative to scaling out—even for very large applications like Wikipedia. We also argue that an additional caching layer for the database is not necessary, as fast in-memory database systems have similar performance to dedicated in-memory caches like Memcached. In addition, a centralized architecture consisting of a single database

server (plus one hot standby for fail over) has the advantage of requiring less hardware and, more importantly, being cheaper to administrate.

To show that this centralized approach is indeed feasible, we demonstrate that a single server running the main memory database system HyPer can handle the database load of Wikipedia—one of the largest web sites in the world. Wikipedia is good case study because the Wikipedia foundation publishes much internal source code and data—including the source code of MediaWiki, the complete history of articles, and (aggregated) server logs. Based on this data, we design the *Monopedia Benchmark*, a database benchmark modeling the traffic of Wikipedia. The Monopedia Benchmark is based on the actual MediaWiki application that drives Wikipedia, but is simplified while still modeling all queries for reading and updating articles, the most frequent actions on the Wikipedia database. The benchmark will be made publicly available and may serve as test bed for future research like comparisons with other database engines and NoSQL approaches [3, 2].

## 2. WIKIPEDIA BENCHMARK

Being ranked 6th globally with 1,689,857 sites linking in according to Alexa Internet [1], Wikipedia is one of the ten most popular web sites in the world and the most popular web site where both the architecture and the traffic data are freely available. Based on the open source architecture of MediaWiki we will run Wikipedia. Additionally, we create *Monopedia Benchmark*, a benchmark derived from the access logs of Wikipedia aimed at testing our architecture.

### 2.1 Current Server Layout

At the start of Wikipedia in 2001, one server sufficed to get access to articles of 4 GB size in total. With increasing popularity, the articles' size increased to more than 314 GiB considering only the English Wikipedia [10]. The current server layout uses six different server clusters at four different locations, of which only two were used for application services including the database, two were used for caching purposes, and the rest for networking [9]. Inside the application services, one master server and a few slave servers for each language specific Wikipedia form the core services using MariaDB databases, the English Wikipedia needs one master server and five slave ones. They provide a search index for the front end and deliver the articles in plain text format to PHP, which renders all macros inside the text to the final HTML representation and finally Apache servers delivers

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>. For any use beyond those covered by this license, obtain permission by emailing [info@vldb.org](mailto:info@vldb.org).

*Proceedings of the VLDB Endowment*, Vol. 10, No. 12  
Copyright 2017 VLDB Endowment 2150-8097/17/08.

these pages with all additional media to the end users. The application on top is MediaWiki, mostly written in PHP, which transforms incoming requests to SQL database queries. Repeated requests to the database will be cached using Memcached, frequently used web pages will be cached by multiple dedicated Squid servers [11].

## 2.2 Centralized Database Architecture

One aim of our project is running Wikipedia on one database server. MediaWiki can be installed using the default schema with MySQL or SQLite. To be used with HyPer, the main memory database system developed at the Technical University of Munich [5], we adapt MediaWiki and the dumps to conform to the PostgreSQL query syntax and its application programming interface. HyPer is running on a separate server. To tap the full potential of the main memory database, articles are always loaded from the database.

## 2.3 Monopedia Benchmark

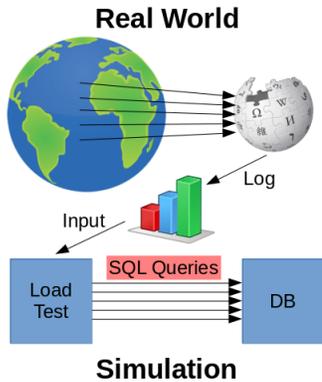


Figure 1: Monopedia Benchmark

The main part of this work is *Monopedia Benchmark* to prove the capability of web scale databases. We develop a load test replaying the database queries of the online encyclopedia Wikipedia (shown in figure 1), in particular every read access and every update of an article. Thus, *Monopedia Benchmark* is the load test simulating concrete traffic of Wikipedia. Creating our benchmark consists of the following steps: extracting relevant database queries and producing the load using a master-worker architecture. In order to replay read and update queries on the articles in the database, we need accurate read and write logs. The read articles can be easily extracted using the pagecount statistics provided by Wikimedia since 2007 [8], aggregated to counts per hour in files like `pageviews-20170331-140000` (for March 31, 2017 at 2 p.m., UTC timezone). The entry for each article in each language consists of the traffic in bytes in the specified hour and of the number of accesses on each article, which can be later used to determine the frequency with which an article will be called by the load test. Based on the write history we determine the frequency for write requests comparable to read statistics.

In the next step, the load test has to simulate user requests, so every database query produced by a request should be imitated. We extract both types of request by sniffing the

```
SELECT page_id, page_namespace,
       page_title, page_restrictions,
       page_is_redirect, page_is_new,
       page_random, page_touched,
       page_links_updated, page_latest,
       page_len, page_content_model
FROM mediawiki.page
WHERE page_namespace = '0' AND
       page_title = $1
LIMIT 1
```

Figure 2: Stored Procedure: Find Page by Title

```
SELECT old_text, old_flags
FROM mediawiki.pagecontent
WHERE old_id = $1
LIMIT 1
```

Figure 3: Stored Procedure: Find Page Content

database log because with this approach it is very unlikely to miss one of the queries spread around in source code. For each incoming read requests among others MediaWiki demands the article's identifier by sending the searched page title first (shown in figure 2), afterwards the content can be called by its identifier (shown in figure 3).

In the last step, we produce the load by a master-worker architecture. Core of the load test is a variable amount of worker threads, flexible in their use to read or update articles. They ensure a high workload and better scalability. Every worker thread chooses an article listed in the pagecount data randomly by draws with replacement and waits for an answer from the database. After a fixed and uniform time span it continues by choosing another article. The load test is a distinct program and can be executed separately from the database. During development, the load test and database can be run on the same machine, but for maximum load benchmarks it is the recommended way to spread them on multiple machines. It can be executed on a remote machine or on the same machine where running the database, communicating either using Unix domain sockets or via TCP. For reproducing *Monopedia Benchmark*, we made the source code<sup>1</sup>, the statistics and the database dumps of the current English Wikipedia from April 1, 2017 available online<sup>2</sup>.

## 2.4 Traffic Data Analysis

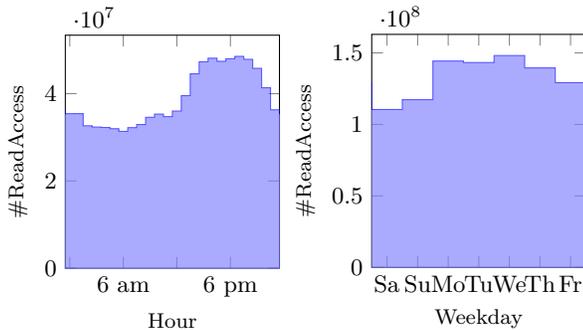
Analyzing the traffic data helps finding the maximal load peak and replaying it. The read analysis refers to the pagecount data of the week of March 25-31, 2017. In this week 932,228,284 English articles were requested to read, most of them from the main namespace containing encyclopedic articles, the rest are project related ones and non existent pages (shown in table 1). We deduct from the articles' history (week of March 25-31, 2017) a total number of 1,143,247 updates per week (16.018 GiB). Taking the average we can deduce a frequency of 1541.38 read requests per second and of 1.89 update requests per second. To show that one database server is enough, it has to beat this frequency. We want to focus our benchmark to find the maximum load. Therefore,

<sup>1</sup><https://dbkemper4.informatik.tu-muenchen.de/monopedia>

<sup>2</sup><http://hyper-db.de/monopedia/>

	#entries	#requests
all	13.3 M	932.2 M
articles	6.9 M	764.1 M
project related	4.7 M	52.0 M
non existent	1.7 M	116.1 M

**Table 1: Summarized Read Traffic Data of English Wikipedia: March 25-31, 2017**

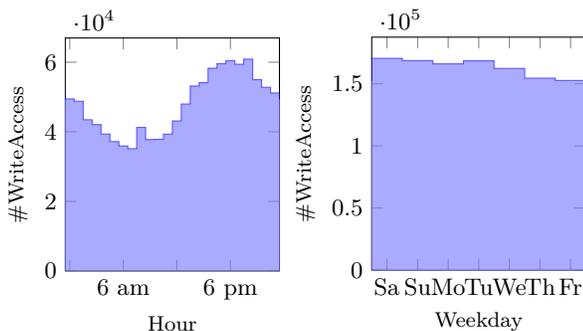


**Figure 4: SELECT SUM(reads) FROM week WHERE date BETWEEN '2017/03/25' AND '2017/03/31'**

we analyze the traffic data aggregated by week and hour (read requests shown in figure 4, update requests shown in figure 5) to find out the traffic peaks and the time-dependent load distribution. When we break down the data into requests per day, we find out the trend to more read requests during the weekdays and a nearly constant number of update requests. The same analysis broken down into requests per hour shows us a load peak between 1 p.m. and 11 p.m. (UTC time zone), which may be caused by US-American Wikipedia users, that are likely the most frequent and numerous users of the English version.

### 3. EVALUATION

We will run six different tests using three different setups. The first test (100%) simulates the current average load of the English Wikipedia (1,541.38 read requests and 1.89 write requests per second) and shows the sufficiency of our architecture handling the current load. In the next step, we will find out possible limits by sending as many requests



**Figure 5: SELECT SUM(writes) FROM week WHERE data BETWEEN '2017/03/25' AND '2017/03/31'**

as the database can handle and counting the highest possible throughput (*no-wait*). Finally, we reduce the number of requests to the highest possible level in respect of the answer's timeliness (*all-on-time*). Like this, we retrieve the highest possible request frequency ensuring timely answers at the same time and we attest our architecture to be ready for managing more requests than the real load. The experiments were run multi-threaded on two machines running both Ubuntu 16.04 and sporting two Intel Xeon E5-2680 v4 CPUs running at 2.40 GHz and 256 GB DDR4 RAM each and connected by a 10 Gbit/s network card. Each CPU has 14 cores and supports Hyper-Threading. Consequently, three setups are possible: in the first setup, both the database and the load test are running on the same machine, communicating via Unix domain sockets; the second setup is like the first, but communicating via TCP, and in the last setup the load test is running remotely, communicating with the database via TCP. An additional script runs the tests automatically. For evaluation, we used the database dumps of the current English Wikipedia from April 1, 2017, the corresponding pageview and writecount statistics from March 31, 2017.

As shown in figure 7 (left), one database server suffices to reply to all incoming read and update requests even without external caching. If we want *all* requests to be processed on time, the main memory database will handle up to 70% more requests (2665 mixed requests per second) than caused by the English Wikipedia's current workload. In summary, the monolithic architecture is able to replace six database servers. Considering just read requests one database can reply to over 150% more load with all requests on time and handles 520% more requests without timing constraints. Under the same constraints, but including write requests, the performance is the same as in *all-in-time*.

As we can see, the monolithic architecture is able to handle more than the current workload. Thereby, we have demonstrated one server using a high-performance database is sufficient to run the 170% of English Wikipedia's load. Considering that during all tests not all available CPUs were used, the network connection seems to be the limiting factor.

### 4. DEMONSTRATION

We created an interactive web interface to control the load test. The interface allows the user to set the access frequency and choose if new workers should read or update articles (shown in figure 6). Having started the load test, the user gets an overview over all worker threads and their current status including the request frequency and duration, the percentage of timeliness requests and the name of the last article accessed. To find out the maximum possible load, he is requested to increase the access frequency. As soon as the requests are no longer on time, the load limit is exceeded. It is a task of the user to work round to the exact limit.

During the demonstration, we will run our architecture based on the English Wikipedia's dump on a server. In parallel, we will demonstrate the load test interface for everyone to reproduce our measurements. We will motivate our demonstration by providing different data files so that the user can replay the traffic of a certain hour of a day. Additionally, everyone is invited to try out different settings (variation of access frequency, number of worker threads) with the aim of exceeding the limits of one main memory database.

## Monopedia *BENCHMARK*

	#req	traffic [MiB]	#err	req/s	KiB/s	OnTime	min	avg	max	last request	title	Rate
Σ Reads	21684	535	0	2041.0	50299.7	96.4%	230μs	1172.3μs	210094μs	1035μs	Thieves'_cant	
Σ Updates	2	0	0	0.0	0.0	100%	14680μs	294773.0μs	574866μs	14680μs	Train	
R1	5409	131	0	507.3	12750.8	95.9%	230μs	1177.7μs	510094μs	1016μs	Piccolo	
R2	5424	134	0	511.3	12234.8	96.6%	253μs	1174.0μs	475277μs	982μs	Pennsylvania	
R3	5425	131	0	513.2	12274.6	97.0%	246μs	1166.4μs	501295μs	1064μs	Paul_Bulcke	
R4	5406	137	0	509.3	13039.5	96.2%	254μs	1171.1μs	316063μs	1035μs	Thieves'_cant	
U1	2	0	0	0.0	0.0	100.0%	14680μs	294773.0μs	574866μs	14680μs	Train	

Figure 6: Snapshot of the web interface

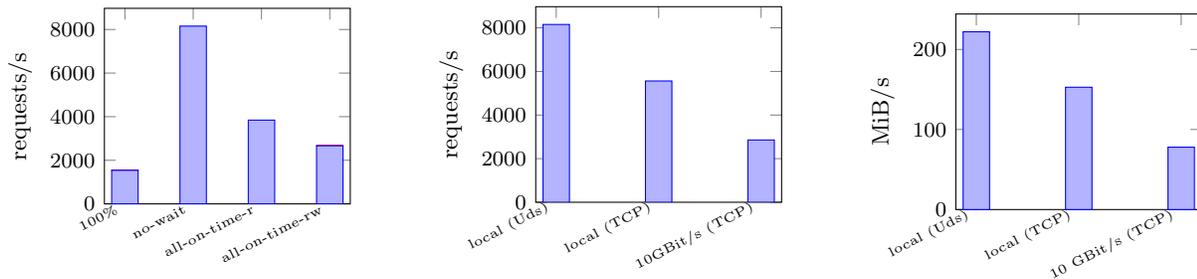


Figure 7: Results: First Tests, Network Test (req/s, MiB/s)

## 5. REFERENCES

- [1] Alexa Internet. wikipedia.org Traffic Statistics. <http://www.alexa.com/siteinfo/wikipedia.org>, 2017. [Online; March 1, 2017].
- [2] A. Dey, A. Fekete, and U. Röhm. Scalable Transactions across Heterogeneous NoSQL Key-Value Data Stores. *Proceedings of the VLDB Endowment*, 6(12):1434–1439, 2013.
- [3] A. Floratou, N. Teletia, D. J. DeWitt, J. M. Patel, and D. Zhang. Can the Elephants Handle the NoSQL Onslaught? *Proceedings of the VLDB Endowment*, 5(12):1712–1723, 2012.
- [4] T. Hunter, T. Moldovan, M. Zaharia, S. Merzgui, J. Ma, M. J. Franklin, P. Abbeel, and A. M. Bayen. Scaling the Mobile Millennium System in the Cloud. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 28. ACM, 2011.
- [5] A. Kemper and T. Neumann. HyPer: A Hybrid OLTP&OLAP Main Memory Database System Based On Virtual Memory Snapshots. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 195–206. IEEE, 2011.
- [6] W.-S. Li, O. Po, W.-P. Hsiung, K. S. Candan, D. Agrawal, Y. Akca, and K. Taniguchi. CachePortal II: Acceleration of Very Large Scale Data Center-Hosted Database-driven Web Applications. pages 1109–1112. VLDB Endowment, 2003.
- [7] D. Porobic, P. Tozun, R. Appuswamy, and A. Ailamaki. More Than A Network: Distributed OLTP on Clusters of Hardware Islands. In *Proceedings of the 12th International Workshop on Data Management on New Hardware*, number EPFL-CONF-218458, 2016.
- [8] Wikimedia. Analytics/data/pagecounts-raw. <https://wikitech.wikimedia.org/wiki/Analytics/Data/Pagecounts-raw>, 2017. [Online; March 1, 2017].
- [9] Wikimedia. Clusters. <https://wikitech.wikimedia.org/wiki/Clusters>, 2017. [Online; March 1, 2017].
- [10] Wikimedia. Downloads. <https://dumps.wikimedia.org/>, 2017. [Online; March 1, 2017].
- [11] Wikimedia. Server. [https://meta.wikimedia.org/wiki/Wikimedia\\_servers](https://meta.wikimedia.org/wiki/Wikimedia_servers), 2017. [Online; March 1, 2017].