Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bachelor's Thesis Nr. 132 b

Systems Group, Department of Computer Science, ETH Zurich

Druid vs. AIM
A comparison of two real-time analytical data storage systems

by

Lukas Striebel

Supervised by

Prof. Donald Kossmann, Lucas Braun, Renato Marroquin

09.02.2015 - 09.07.2015

inf | Informatik
Computer Science

Abstract

In today's society, data is arguably one of the most valuable goods. In recent years, many data related research fields have thrived: Big data and advanced analytics, health IT and Bioinformatics, data protection and encryption, just to name a few. Dealing with huge amount of data is without doubt essential for any company. In this thesis two different data stores, AIM and Druid, are compared against each other. The thesis further explores how these two system could be used together in order to support each other. Our experiments have shown promising results and suggest that Druid may indeed serve as a proper backup system for AIM.

Contents

1 Introduction **3**
1.1 Motivation . . . 3
1.2 Thesis Structure . . . 3

2 Background **4**
2.1 The AIM Architecture . . . 4
2.2 The Druid Architecture . . . 6
2.3 The original Benchmark . . . 7

3 Methods **8**
3.1 First Attempts . . . 8
3.2 Rethinking the benchmark . . . 8

4 Results and Discussion **10**
4.1 Small Benchmark . . . 10
4.2 Hadoop for Big Data . . . 13

5 Summary **14**
5.1 Conclusion . . . 14
5.2 Acknowledgements . . . 14

1 Introduction

1.1 Motivation

As of 2015, the number of mobile devices has officially surpassed the world's population [6]. Billions of phone calls and text messages are sent every day. Mobile phones have become an essential part of daily routine. This development has lead to numerous telecommunication companies sprouting and prospering. These companies are in constant competition to provide the best possible service to their customers, as well as maintaining high profits.

One of the many challenges those companies have to face, is to employ a suitable storage system for their users' data and call histories. Scalability and efficiency are only a few of the countless requirements such a system must fulfill. Data stores which are capable of dealing with these difficulties are in high demand. The goal of this thesis is compare two different databases, Druid and AIM, in a realistic environment. We will further lay out how the two system can be used together in order to complement each other and cover each other's weaknesses.

1.2 Thesis Structure

The thesis is structured as follows. In Section 2 the Druid and AIM systems are introduced and the benchmark is explained. Section 3 contains the different approaches, some successful and some unsuccessful, on how to run this benchmark on Druid. The obtained results are discussed in section 4. Section 5 summarizes my conclusions and provides an outlook.

2 Background

2.1 The AIM Architecture

The AIM system [2] is currently being developed at ETHZ by various people. It is designed for storing large amounts of data as well as dealing with synchronous updates and queries. A possible use case would e.g. be a telecommunications company where telephone calls would constitute the database entries. Queries would be needed to retrieve specified call statistics for individual customers or customer groups, e.g. average call duration, number of calls within a specified period, and so on. A basic overview of the AIM architecture is displayed in Figure 1.

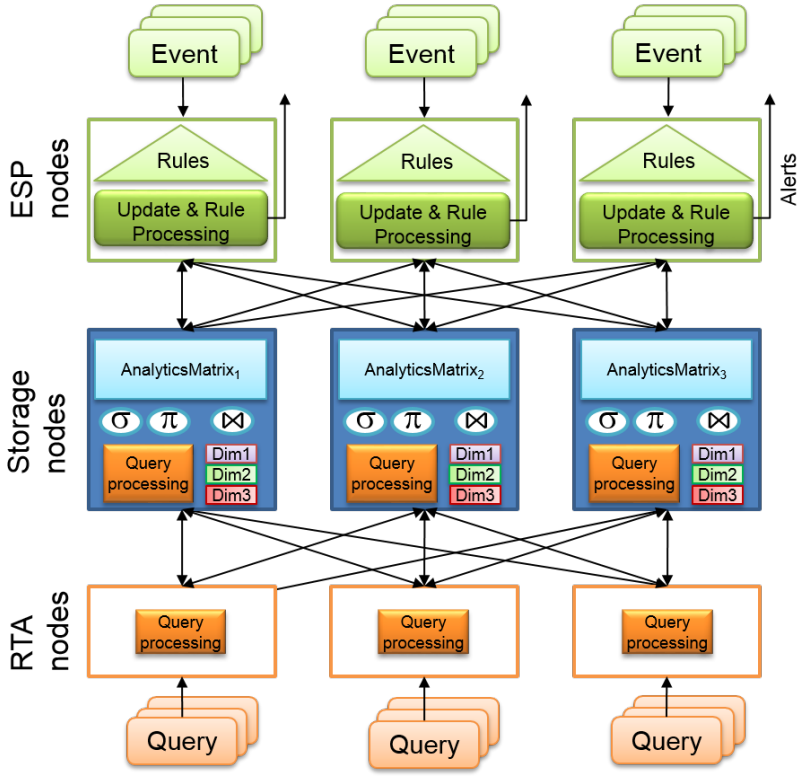

Figure 1: The AIM architecture, graphic taken from [2]

The core idea behind AIM is to have two separate parts, the SEP and the RTA, to handle the different challenges. Both parts operate on a large amount of data, called the *Analytics Matrix*, which can be imagined as a very large and wide table. AIM uses a shared nothing architecture and stores this Analytics Matrix in column oriented memory structure called *ColumnMap*.

The *Stream Event Processing* (SEP) receives updates and modifies the Analytics Matrix accordingly. Incoming updates are stored inside a buffer, which periodically merges into main storage through a mechanism called *Differential Updates*. Every time an update is executed, a set of *Business Rules* is evaluated, checking if a customer is eligible for certain special offers. For example, a company might offer specific discounts for highly active users.

Due to its size, the Analytics Matrix is usually not stored on a single machine, but instead distributed amongst multiple data storage nodes. The *Real-Time Analytics Processing* (RTA) is responsible for processing the queries and assembling the partial results received from the various distributed data storage nodes.

2.2 The Druid Architecture

Developed by Metamarkets Inc., the Druid system [1] is built on four different types of nodes each of which fulfilling one of these tasks: Data ingesting, storing, querying and coordinating. Both the Overlord node and the Realtime node are responsible for the data ingestion. The difference between them is that the Realtime node is used in case of a continuous data stream, e.g. Amazon sales whereas the Overlord node is used whenever the data to be ingested is already known, as is the case in the AIM benchmark. The Historical node saves the ingested data segments on Deep Storage, which can be a hard disk, Amazon S3 or whatever else is specified.
The Broker node directs incoming queries to the corresponding Historical node.
The Coordinator nodes in combination with Zookeeper [5] manages the interactions between the different nodes.

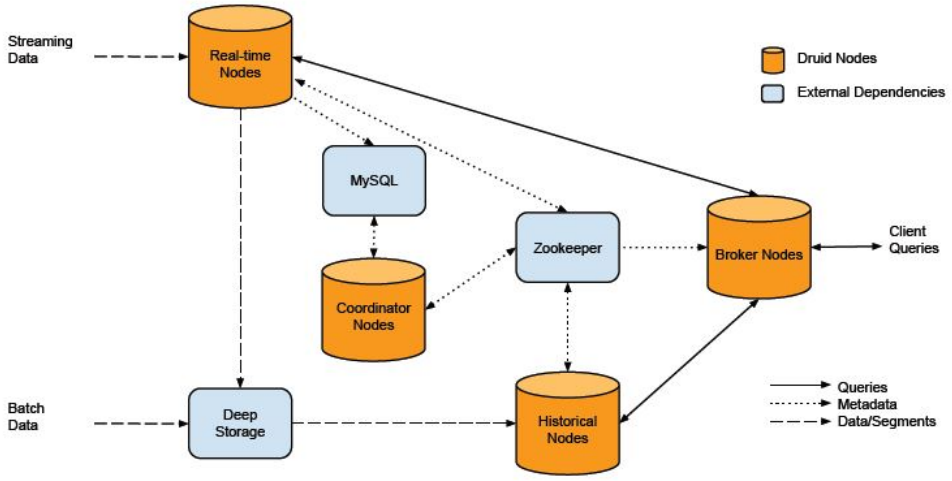

Figure 2: The Druid architecture, graphic taken from [1]

There are a number of extensions compatible with Druid, most notably Apache Hadoop [4] which will be used later on in this thesis. Hadoop on its own is a powerful distributed data storage and processing framework. However in our case we will only use it to replace the Overlord node in our Druid system. As stated by the developers and verified by personal experience, the Overlord node works efficiently only as long as the size of the data file is below 1 GB. After exceeding this threshold, performance drops significantly. We therefore used a single Hadoop node for the ingestion of larger files.

Druid is open source and currently available at [3].

2.3 The original Benchmark

For the benchmark, we were using the mentioned example of a large telephone company with millions of subscribers, also referred to as customers. The original benchmark consisted of two parts, one for the RTA and another for the SEP. The Analytics Matrix was set up to contain a total of ten million subscribers. Each customer was described with 42 unique attributes such as number of calls or longest call duration. These 42 attributes were duplicated 13 times in order to create a more realistic and challenging environment. This resulted in a total amount of 546 attributes per subscriber.

In the first part of the benchmark, the performance of the SEP was evaluated. Three million updates, each one relating to a single subscriber, were applied to the Analytics Matrix and the elapsed time was measured.

In the RTA part, the performance for answering seven different SQL queries, featuring various Joins and Aggregations, was measured.

3 Methods

3.1 First Attempts

In our first attempts we tried to implement exactly the same AIM benchmark on Druid. This meant that Druid had to store only one row for each customer, which contained various static attributes such as subscriber-ID, cheapest call this week etc., as well as the entire 546 aggregations, resulting in a very wide table. However, it also yielded the benefit of having a table of constant size, rather than an ever growing one. Approaching the benchmark this way greatly simplified the RTA part. Since Druid does not understand SQL, the only thing to do was to translate the original seven queries into a format that Druid was able to understand.

However, it soon became clear that updating existing data in Druid is quite difficult, since Druid does not feature any kind of native updating functionality. Instead it is suggested to simply ingest a new, more up-to-date record which would overwrite the existing one. Therefore the only possible way to execute an update in the sense of the SEP benchmark would be to send a query to retrieve the old record, manually update it and then re-ingest that record. Unsurprisingly, this approach performed very poorly. As we did not find a more elegant solution to update the records in Druid, we decided to abandon this original benchmark and to take a different path.

3.2 Rethinking the benchmark

As AIM lacks a reliable recovery system, we decided to instead test whether Druid could be used as a suitable backup system for AIM. This implied that Druid now was used to save all the calls rather than all the subscribers. In case of a critical failure of AIM, would it be possible to reconstruct the customers' call data using the Druid backup, in a reasonable time period?

Storing the calls instead of the users' data presented a set of new challenges. One of our main concerns was scalability, since the number of calls increases substantially faster than the number of subscribers. Would Druid be able to deal with a such rapidly growing database?

We decided to focus on three major points in our benchmark:

- The **ingestion rate**, measured in rows/second and kB/second
- The time it takes to restore all attributes for every customer (**full restore**)
- The time it takes to restore all attributes for a single customer (**single restore**)

We set up our experiments to verify that these values stayed within an acceptable magnitude, even if the number of calls grows very large. In order to prove this point, we decided to let Druid consecutively ingest 10 segments of data, and send a set of queries after each data segment. Every segment was 120 MB large and contained 1'048'576 rows of phone calls. Each phone call was described using five attributes:

- The time stamp
- The subscriber-ID, ranging from 1 to 1'000'000
- The call duration, ranging from 1 to 10'000
- The cost of the call, ranging from 1.0 to 99.9
- A boolean to indicate whether the call was local or non-local

Each query would then reconstruct one of the original 42 attributes, e.g. the sum of a subscriber's call durations, the longest call this week etc. Measuring the ingestion rate as well as the time it takes for these queries to complete should give us an idea whether or not Druid is eligible to act as a backup system for AIM.

4 Results and Discussion

4.1 Small Benchmark

As mention in Section 3, the three domains under measurement were the ingestion rate, the full restore and the single restore. However, we were more interested in their development over the course of the benchmark rather than the actual values. As the database grew larger, from 1 million rows after the first segment up to 10 millions after the last one, the amount of time for a full restore was expected to increase, whereas the time for the ingestion and the single restore should stay constant to some degree.

The first successful benchmarks were run on a private machine with 7.8 GB of memory and an Intel Core i7-2600 CPU @3.4 GHz x 8, using Ubuntu 14.04 LTS 64-bit. (See Figure 3)

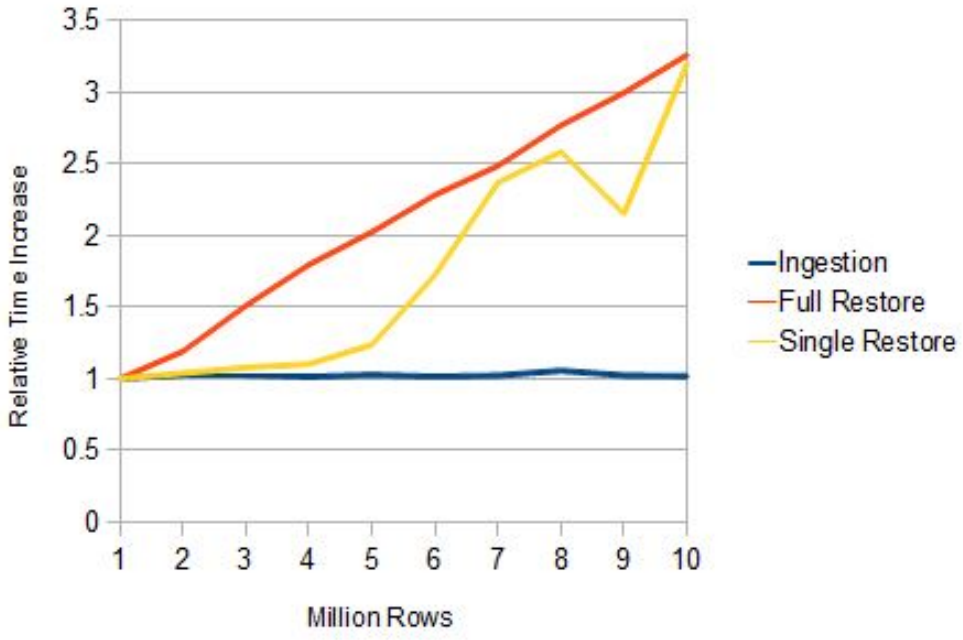

Figure 3: Small Benchmark on a private machine

In this diagram, the relative time in- or decrease over the course of the benchmark **in relation to the very first segment**, is displayed. The development shows a nearly constant ingestion rate, as expected. Ingesting a single segments took 124 seconds on average, resulting in a rate of 8460 rows/second or 947 kB/second. These results were unsurprising and met the expectations.

However, the single restore behaved oddly. Later experiments indicate that the results likely were disturbed by extreme outliers. The results of the subsequent benchmarks support this claim.

Unfortunately, Druid was not able to restore the full Analytics matrix in a single query. All such attempts ended in a time-out failure. In order to overcome this problem, we split the subscribers in 10 smaller groups, according to their IDs. This step resolved the problem and we were able to obtain reasonable results. As the graphic shows the query times increased only linearly and not exponentially. After the 10th segment, the full restore was 3.5 times slower than after the first. These times were acceptable, considering the amount of data had increased by a factor of 10.

The same benchmark was also run on Euler07, a machine with 129 GB RAM and an Intel Xeon CPU E5-2609 @ 2.40GHz x 8, using a Linux kernel 3.18.
These following results were obtained. (See Figure 4)

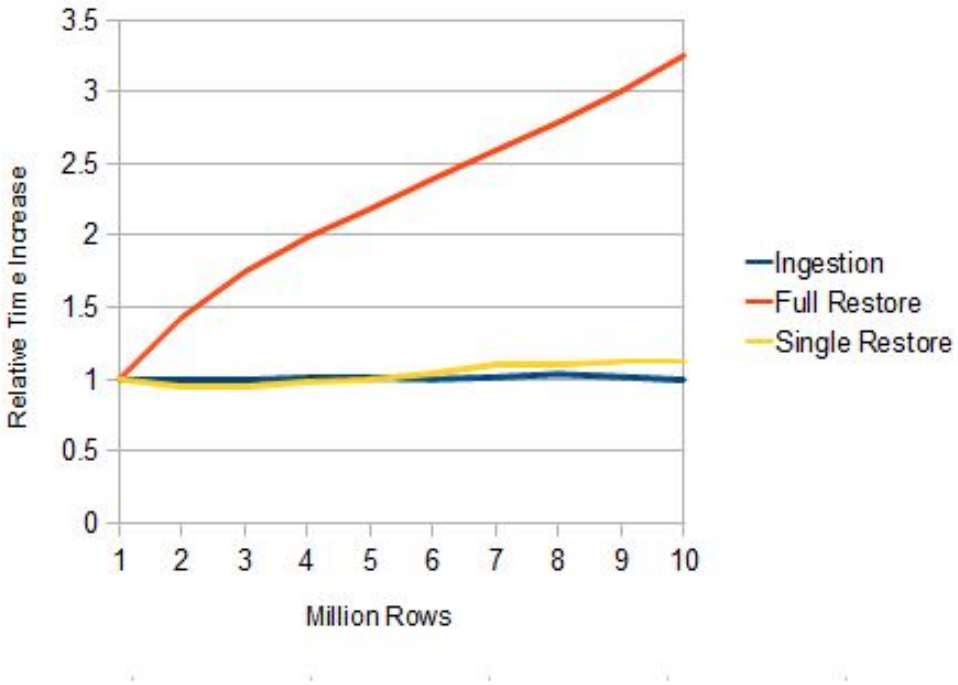

Figure 4: Small Benchmark on the Euler07

Again, the diagram depicts the relative time in- or decreases of the 3 fields under measurement. The ingestion rate and the full restore behaved almost identical to the other benchmark, remaining constant resp. increasing up to a factor 3.5.

Despite the enormous gain of memory, the ingestion rate did not improve. It actually decreased down to 6'520 rows/second resp. 747 kB/second. This indicates that the ingestion was capped by the CPU, which is not surprising.

On the other hand, the absolute times for the full restore decreased significantly, as apparently querying is capped by memory.

The interesting point to notice is the single record restore, which remained constant, in contrast to the previous benchmark. The absolute times for the single restore after the first segment remained unchanged in comparison to the other benchmark.

4.2 Hadoop for Big Data

In order to create a more realistic test environment, we increased the size of each data segment by a factor 10. The separate files now contained 10'485'760 rows each and were 1.2 GB large. In order to ingest these large datasets, we set up a single Hadoop node, rather than relying on the Overlord node. Besides these changes the benchmark remained the same. The following results were obtained on the same private machine as in section 4.1. (See Figure 5)

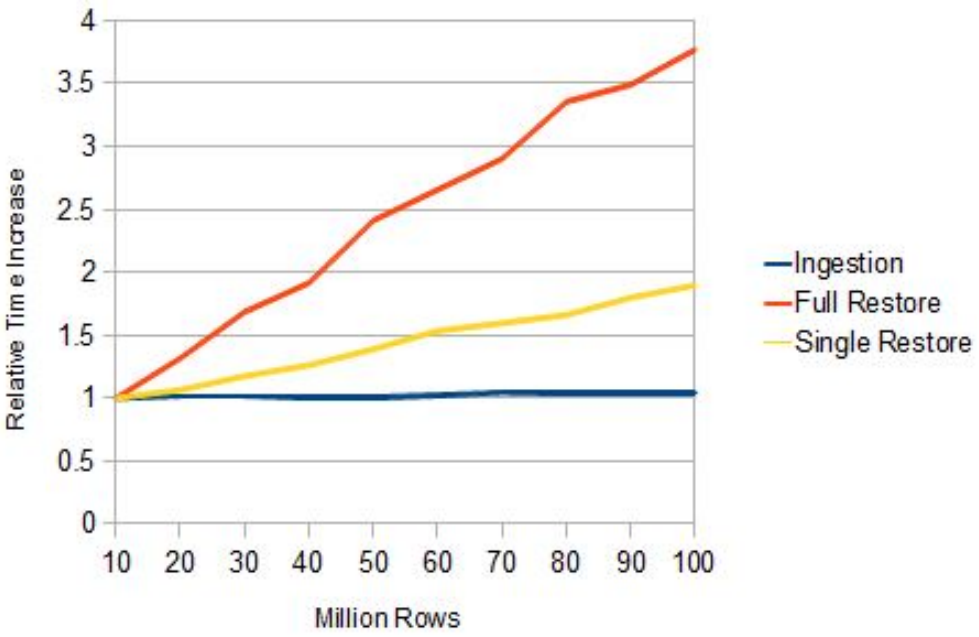

Figure 5: Large Benchmark on a private machine

Once again, the ingestion rate remained more or less constant throughout the entire benchmark. It took Druid resp. Hadoop 689 seconds on average to complete the task, which results in an ingestion rate of 15'200 rows/second or 1'740 kB/second. The development of the query times is very similar to the small benchmark on the Euler machine, although they executed a lot slower due to the increased size of the database. The elapsed times for the full restore increase linearly up to a factor 4. The single restore slows down slightly, up to a factor 2.

In all three benchmarks the initial ingestion rate remained constant and the query times increased only linearly, up to a factor 4 for the full restore resp. a factor 2 for the single restore. These results met the expectations and proved the scalability of our Druid-AIM setup.

5 Summary

5.1 Conclusion

Motivated by use cases from the telecommunications industry, we attempted to put the performance of two promising data stores, Druid and AIM, into perspective. Having already a realistic and challenging test environment for AIM at hand, we tried to execute the exact same benchmark on Druid. However, it soon became clear that such a comparison makes little sense, since the two systems are designed for completely different purposes. Druid excels at ingestion of a huge continuous stream of data while having trouble updating existing data, which is one of the core strengths of AIM.

Rather than heavily modifying the AIM benchmark, we came up with a new one which was supposed to verify Druid's suitability as a backup store for AIM, as AIM currently lacks the ability to restore its content in case of severe failure. The obtained results indicate that this is indeed the case, as Druid's ingestion rate remained constant, and the query times increased only linearly which is acceptable.

The original goal of this thesis was to execute the AIM-benchmark on Druid, which we unfortunately were unable to achieve. However, we were able to demonstrate that the collaboration of the two system shows promising results. It remains to be seen whether Druid and AIM are able to assert themselves in the huge and competitive field of data stores. There are countless competitors, such as Impala, Cassandra or Hadoop, and it would be interesting to see how they fare in direct comparison. However, this is material for future work.

5.2 Acknowledgements

I would like to thank my supervisors Lucas Braun and Renato Marroquín for their continuous support and help. They provided many meaningful suggestions and proposals. Their expertise and experience was invaluable.

I would also like to thank Professor Kossmann for taking the time to supervise my thesis.

A special thanks thanks goes to my parents for keeping me motivated and for reviewing this thesis.

References

[1] Fangjin Yang, Eric Tschetter, Xavier Léauté, Nelson Ray, Gian Merlino, Deep Ganguli. *Druid. A Real-time Analytical Data Store*, In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, pp. 251 - 264

[2] Lucas Braun, Thomas Etter, Georgios Gasparis, Martin Kaufmann, Donald Kossmann, Daniel Widmer, Aharon Avitzur, Anthony Iliopoulos, Eliezer Levy, Ning Liang. *Analytics in Motion. High Performance Event-Processing AND Real-Time Analytics in the Same Database*, In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 157 - 168

[3] Metamarkets Inc. *Druid.* http://druid.io/, 2014. [Online; accessed July 2015].

[4] Apache Foundation. *Hadoop.* http://hadoop.apache.org/, 2013. [Online; accessed July 2015].

[5] Apache Foundation. *Zookeeper.* http://zookeeper.apache.org/, 2014. [Online; accessed July 2015].

[6] The Independent. *There are officially more mobile devices than people in the world.* http://www.independent.co.uk/life-style/gadgets-and-tech/news/there-are-officially-more-mobile-devices-than-people-in-the-world-9780518.html, 2014. [Online; accessed July 2015].