Avatara: OLAP for Web-scale Analytics Products

Lili Wu Roshan Sumbaly Chris Riccomini Gordon Koo Hyung Jin Kim Jay Kreps Sam Shah
LinkedIn
{lwu,rsumbaly,criccomini,gkoo,ekim,jkreps,samshah}@linkedin.com

ABSTRACT

Multidimensional data generated by members on websites has seen massive growth in recent years. OLAP is a well-suited solution for mining and analyzing this data. Providing insights derived from this analysis has become crucial for these websites to give members greater value. For example, LinkedIn, the largest professional social network, provides its professional members rich analytics features like "Who's Viewed My Profile?" and "Who's Viewed This Job?" The data behind these features form cubes that must be efficiently served at scale, and can be neatly sharded to do so. To serve our growing 160 million member base, we built a scalable and fast OLAP serving system called Avatara to solve this *many, small cubes* problem. At LinkedIn, Avatara has been powering several analytics features on the site for the past two years.

1. INTRODUCTION

LinkedIn, the largest professional social network on the Internet, has more than 160 million members. Our members create profiles containing rich structured data such as their industry, work experience, and skills. By coupling this multidimensional profile information with member activity data, we can provide various data-derived insights to our members. Two examples of analytics features are *Who's Viewed My Profile?* (WVMP) as shown in Figure 1a, which provides analytics around members who viewed your profile, and *Who's Viewed This Job?* (WVTJ) as shown in Figure 1b, which provides aggregate analytics to recruiters on members viewing their posted jobs. These features pre-aggregate data to generate reports from different dimensions. For example, WVTJ shows the number of job views broken down by time, title, and company. OLAP is an ideal solution to handle these tasks.

There are several existing solutions in the market for enterprise users. These enterprise solutions are designed for offline analytics and traditional data warehousing, and do not efficiently support the high throughput, low latency, and high availability needs of a high-traffic website. That is, these results must be presented to the user on page load, requiring response times in tens of milliseconds.

In these use cases, queries are mostly known *a priori* because the product interface limits the user to select some subset of known queries. This means expensive operations like joins can be precomputed. Additionally, queries span relatively few—usually tens to at most a hundred—dimensions. These queries, and consequently, the cubes, can be sharded across a primary dimension. For WVMP, we can shard the cube by member id, as the product does not allow analyzing profile views of anyone other than the member currently logged in. Similarly, for WVTJ, the cube can be sharded by the job itself because queries do not span across jobs. Each shard contains a relatively small amount of data (a few megabytes, up to tens of megabytes), though there are many shards (160 million, one for each LinkedIn member, for WVMP). We call this class of OLAP the *many, small cubes* scenario.

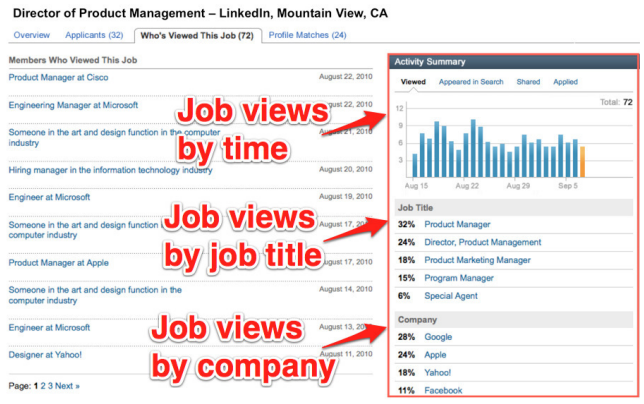
Traditional distributed OLAP systems leverage the scatter-gather paradigm [1] to retrieve and consolidate partial results from data partitions spread across nodes. An OLAP operation cannot complete until all the nodes have responded, resulting in slow response times when any one node is overloaded [2]. This is undesirable in a web serving scenario where response times need to be predictable.

This paradigm can also affect the availability of the system. Consider a scenario where on a cluster of 100 nodes, a failure of 10 minutes for each machine is expected every 30 days. The availability of the system is: $(1 - \text{MTTR}/\text{MTTF})^n$, where MTTR is the mean time to recovery, MTTF is the mean time to failure, and n is the number of nodes required to satisfy the query. In our example, the availability of a query is: $(1 - 10/(30 \cdot 24 \cdot 60))^{100} = 0.977115$. This may initially seem like a sufficiently available system, but it means that a recruiter visiting WVTJ three times a day to check on the status of their job posting would see around 25 "not available" errors in a year. This translates to two errors every month, which is not a pleasant user experience. However, if the query could be satisfied by cube operations within a shard, we can colocate the cube such that every query only requires a single disk fetch. This sharding approach increases the availability of the system to $1-10/(30\cdot24\cdot60) = 0.999768$, which, for the recruiter, translates to 0.25 "not available" errors per year—fairly reasonable. There are several assumptions and simplifications here, but this need forms the basis for highly-available key-value systems, such as Amazon's Dynamo [5].

In this work, we present *Avatara*, a fast, scalable OLAP system to handle the many, small cubes scenario we encounter with our various analytics features. The system provides a simple, expressive grammar for application developers to construct cubes and query them at scale. The sharding of a cube dimension fits well into a key-value model, allowing us to leverage a distributed key-value store, such as Voldemort [10], to provide low latency, highly available access to cubes. On query, a small cube is streamed from disk with the predicate computed at the storage layer. In addition, Avatara leverages an offline elastic computing infrastructure, such as

Proceedings of the VLDB Endowment, Vol. 5, No. 12

(a)



(b)

Figure 1: Two products using Avatara: (a) *Who's Viewed My Profile?* provides rich analytics around people who viewed your profile, including profile views by time, industry, and country; (b) *Who's Viewed This Job?*, provides analytics around members who viewed a posted job, including job views by time, job title, and company.

Hadoop [11], to precompute its cubes and perform joins outside of the serving system. These cubes are bulk loaded into the serving system periodically, usually every couple of hours. All components of this system are horizontally scalable, making it easy to add capacity without downtime.

This system has been in use at LinkedIn for the past two years and powers several analytics features on the site.

2. RELATED WORK

Distributed data warehouses allow users to run OLAP queries across various nodes. In general, request latencies to these warehouses are heavily dependent on the slowest node's response as the data is spread on various nodes. Avatara stores a cube's data

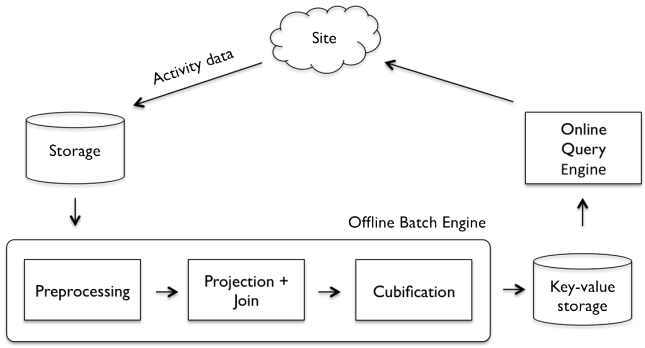


Figure 2: Avatara architecture consists of an offline batch engine, which runs Hadoop jobs to transform the data to cubes, and an online query engine, which fetches results from a key-value store.

in one location so that retrieval only requires a single disk fetch. Avatara also leverages some performance optimizations [1, 2], particularly pushing down predicates to the storage layer, to improve the response time of queries.

MR-Cube [8] efficiently materializes cubes for holistic measures while fully using the parallelism of a MapReduce [4] framework. However, the system does not provide a query engine that can respond to user requests at page load. Avatara emphasizes request latency as it caters to web-scale traffic. It achieves this by sharding on a primary key of the dataset to turn a single big cube into many small cubes.

Avatara also leverages the well-studied key-value paradigm used by various serving systems for storing data. Some examples of low latency, high availability applications using this paradigm include Amazon's shopping cart powered by Dynamo [5], Yahoo's user database powered by PNUTS [3], and LinkedIn's data features [10].

3. ARCHITECTURE

An OLAP system generally consists of two subsystems: cube computation and query serving. Most enterprise solutions couple the two together. Avatara, on the other hand, is an offline/online system that provides high throughput during batch cube computation and low latency during online query serving.

As shown in Figure 2, Avatara has two components: an offline batch engine and an online query engine. We observe that for our use cases, while the online query serving needs to be fast, we can currently tolerate data that is stale by a couple of hours. For example, when a job view occurs, it does not need to be reflected immediately in the WVTJ dashboard.

The offline batch engine performs user-defined joins and pre-aggregation on activity streams, resulting in sharded small cubes. We use Hadoop [11] to power our batch engine, leveraging its built-in fault tolerance and horizontal scalability. The computed cubes are then bulk loaded into a distributed key-value store for fast online serving.

Clients issue SQL-like queries to an online query engine, which then pushes down the computation (such as filtering, sorting, and aggregation operations) to the key-value store. Our architecture is pluggable and can support any key-value store. At LinkedIn, we use Voldemort [10].

Both offline and online engines have the capability to materialize cubes, providing application developers the choice to trade off between faster response time or more flexible query patterns. We highlight the choice we adopted for our products in Section 3.4. The rest of this section illustrates the architecture of Avatara with a running example of WVMP (Figure 1a).

```
input.profile_views=/profile_views
input.member_info=/member_info

// pre-processing
time.column=profile_views.time
time.aggregation_level=weekly


// dimensions and fact tables
dimensions=
 member_info.member_id,
 member_info.industry,
 member_info.country

facts=
 profile_views.viewee_id,
 profile_views.viewer_id,
 profile_views.time

measure=profile_views.visit

join=
 profile_views.viewer_id,
 member_info.member_id


cube.name=wvmp-cube-profile-views
cube.shard_key=profile_views.viewee_id
```

(1) → Preprocessing

(2) → Projection & Join

(3) → Cubification

Figure 3: Cube configuration file of the offline batch engine for the *Who's Viewed My Profile?* product. The system builds a cube of profile view visits by time, industry, and country.

```
{
   dimensions:  {
     viewer_id:          1,
     viewer_industry:   87,
     viewer_country:    us,
     view_time:         1330214400
   },
   measures:  {
     visit:             3
   }
}
```

Figure 4: The resulting cube storage structure for the *Who's Viewed My Profile?* cube definition from Figure 3. In Avatara, cubes are stored as multidimensional arrays.

```
AvataraQuery query = new AvataraSqlishBuilder()
   .setCube("wvmp-cube-profile-views")
   .setShardKey(member_id)
   .select("visit")
   .select("member_info.industry")
   .group("member_info.industry")
   .sum("visit")
   .order("visit", "desc")
   .limit(10)
   .build();

AvataraResult result = queryEngine.getCube(query);
```

Figure 5: An example client query to retrieve the top 10 profile view counts by industry from the *Who's Viewed My Profile?* cube for a given member.

3.1 Offline Batch Engine

The batch processing pipeline (Figure 2) consists of three phases: preprocessing, projections and joins, and cubification. Each phase produces output that is the input for the subsequent phase. All of these phases are driven by a simple configuration file written by an application developer, who can be agnostic to how the system works.

Figure 3 shows the configuration for WVMP. The first phase shows input file paths for profile view events and member information. At this stage, the developer can perform any necessary preprocessing on the input to prepare the data. There are several built-in preprocessing functions available for common use cases. This example uses a built-in function to roll up a time column to "week" granularity. The developer can also specify a customized script for further processing. The output of this phase is saved in a temporary location that serves as the input to the next phase in our pipeline.

The second phase models the preprocessed data as a star-schema [7], and builds the dimension and fact tables. For our WVMP example, the dimension table has the industry and country details of the viewer, while the fact table has details of the profile view event, such as viewing timestamp and member identifiers of both the viewer and the viewee. The profile view visit count is the measure in the fact table. A join key ties the dimension and fact tables together; WVMP joins the tables using the viewer's member id. The offline engine projects and joins the fields, then outputs the results to another temporary location.

The final phase is cubification. This phase partitions the data by the cube shard key and generates small cubes. Avatara provides the flexibility to pre-aggregate these cubes based on certain dimensions. The cube is a multidimensional OLAP (MOLAP) [9] formatted blob combining dimensions and measures. This format uses multidimensional arrays, providing optimized storage: the query engine can retrieve this data in a single disk fetch, resulting in fast response time. Here, the data is structured as an array of tuples of the form $(d_1, d_2, \ldots, d_n, m_1, m_2, \ldots, m_m)$, where d_i represents a dimension and m_j represents a measure. For example, Figure 4 shows the layout for the WVMP cube. The first section highlights the dimensions of the viewer including member id, industry, country, and the week viewing activity happened (our data is rolled up in the "week" granularity). The second section shows the measures: in WVMP, we track the profile view visit count of how many times the viewer visited the viewee's profile in a given week. These resulting cubes are then bulk loaded into a distributed key-value store, with the key being the cube shard key. In WVMP, it is the viewee's member id.

The MOLAP format we adopted is compact, but has the disadvantage of being inflexible in the evolution of dimensions, such as adding a new dimension. Due to our requirement of low latency during online serving, we mandate joins to only happen offline. This also means that the dimensions of a cube must be identified *a priori*, which is not a problem as our user interfaces do not change dynamically. Further, if required, the bulk loading nature of Avatara allows the developer to rebuild and repush the full dataset with a simple configuration change.

The offline engine processes data incrementally by storing results from previous runs. This allows processing to happen multiple times per day.

3.2 Online Query Engine

Avatara's query engine retrieves and processes data from the key-value store before returning results to clients. Because we chose to build a compact cube per sharded key, the retrieval step is quite fast. The query engine models a SQL-like syntax and supports operations such as select, where, group-by, having, order, limit, and basic math operations (count, percent, sum, and average). The wide-spread adoption of SQL makes it easy for application developers to interact with cubes. The query engine is stateless, allowing the system to easily scale horizontally.

Figure 5 shows the query to compute the top 10 profile visit counts by industry for the WVMP application: select the "visits" and "industry" fields, group visits by "industry", and perform a sum of visits. Finally, the results are sorted in descending order of visit count and are limited to 10 rows for display. Similarly, to display profile view by country, the application developer selects the "country" field without recomputing any cubes.

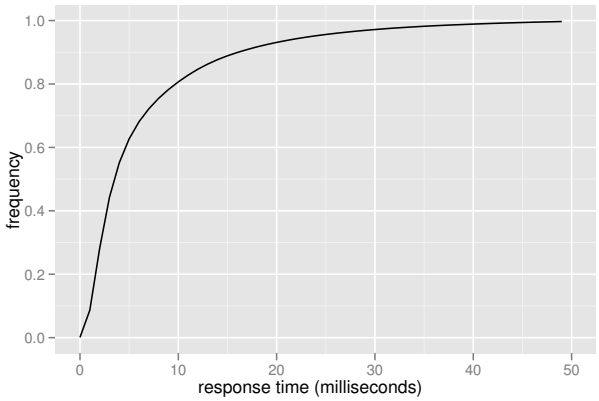


Figure 6: Query latency CDF for a high-traffic day. 95% of queries can be served within 25 ms.

If the key-value store supports local storage node computation, the query engine can push down the predicate to decrease data transfer. This is straightforward as it involves performing projections and rollups on a single blob of cube data. A concern might be that local processing could add additional load on each storage node, but we have found that in the case of Voldemort, the storage nodes are I/O-bound with ample free CPU cycles for computation.

3.3 Cube Thinning

Most cubes can easily be processed by Avatara, but due to the power-law nature of the Internet [6], there will be a few cubes that are too large to process with an acceptable amount of frontend latency. For example, President Barack Obama is an active member of the site and his profile is viewed several orders of magnitude more than most other members. This makes the WVMP cube for him prohibitively large to process at page load. To deal with these rare, large cubes, Avatara provides a mechanism for developers to set priorities and constraints on dimensions that can be aggregated to a particular value (such as an "other" category). The system also has the ability to drop data across predefined dimensions. For example, WVMP can choose to drop data across the time dimension, resulting in a shorter history for these heavy-hitters.

3.4 Discussion

Avatara provides flexibility in terms of where cube materialization can happen: operations such as sum, average, order, or limit can be performed offline or online. With more offline aggregation, online queries will be faster, but naturally less flexible. For example, after a developer specifies the granularity of a time dimension to be at the "week" level in the offline phase, future online aggregations can only happen at equal or coarser levels such as "weeks" or "months". For WVMP, partial materialization happens offline to roll up profile visits into a weekly aggregation level; the remaining materialization happens online because the cubes are small. This also enables us to introduce new query patterns by selecting a different set of dimensions without recomputing any cubes.

4. PRODUCTION WORKLOADS

Avatara has been running successfully at LinkedIn for more than two years. It powers some of our larger web-scale applications such as "Who's Viewed My Profile?", "Who's Viewed This Job?", "Jobs You May Be Interested In", and more. The single configuration file model and SQL-like interaction with the query engine are easy for any application developer to understand and has enabled multiple teams at LinkedIn to provide useful insights to our members with quick development cycles. Avatara uses Hadoop [11] as its batch computing infrastructure, and Voldemort batch extensions [10] for efficient bulk loading from Hadoop as its key-value storage layer. The Hadoop batch engine can handle terabytes of input data with a turnaround time of hours. On the other hand, Voldemort as the key-value store behind the query engine responds to client queries in milliseconds.

Figure 6 shows the query latency for a high-traffic day for LinkedIn's "Who's Viewed My Profile?" feature. As the figure shows, 80% of queries can be satisfied within 10 ms with over 90% of queries returning within 20 ms.

5. CONCLUSION AND FUTURE WORK

Avatara, the in-house OLAP system at LinkedIn, provides a generic batch processing platform, allowing any developer to build OLAP cubes with a single configuration file. The system uses a hybrid offline/online strategy coupled with sharding into a key-value store by an application-specified primary dimension to support OLAP queries at web scale.

We are working on adding near-line updates to cubes to support applications with stricter data refresh requirements. Some of the challenges include read scaling in the presence of writes, memory-limited execution of windowed joins, and multitenant issues such as those introduced by multiple colocated streams and the execution of assorted user-defined functions.

6. REFERENCES

[1] M. O. Akinde, M. H. Böhlen, T. Johnson, L. V. S. Lakshmanan, and D. Srivastava. Efficient OLAP Query Processing in Distributed Data Warehouses. *Information Systems*, 28(1-2):111–135, 2003.

[2] R. Almeida, J. Vieira, M. Vieira, H. Madeira, and J. Bernardino. Efficient Data Distribution for DWS. In *DaWaK*, pages 75–86, 2008.

[3] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. PNUTS: Yahoo!'s Hosted Data Serving Platform. *PVLDB*, 1(2):1277–1288, 2008.

[4] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, pages 137–150, 2004.

[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's Highly Available Key-Value Store. *SIGOPS Operating Systems Review*, 41(6):205–220, 2007.

[6] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-Law Relationships of the Internet Topology. In *SIGCOMM*, pages 251–262, 1999.

[7] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, Inc., 2nd edition, 2002.

[8] A. Nandi, C. Yu, P. Bohannon, and R. Ramakrishnan. Distributed Cube Materialization on Holistic Measures. In *ICDE*, pages 183–194, 2011.

[9] T. B. Pedersen and C. S. Jensen. Multidimensional Database Technology. *Computer*, 34(12):40–46, 2001.

[10] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah. Serving Large-scale Batch Computed Data with Project Voldemort. In *FAST*, pages 223–235, 2012.

[11] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, 1st edition, 2009.