Visual Data Management System

github.com/IntelLabs/vdms
aperturedata.io

Vishakha Gupta, ApertureData Inc.
Luis Remis, Intel Labs
Typical Machine Learning (ML) Workload

Visual Workload: Metadata + Visual Data
Typical Machine Learning (ML) Workload

Visual Workload: Metadata + Visual Data

Metadata -> Relational Database, Graph Database
Typical Machine Learning (ML) Workload

Visual Workload: Metadata + Visual Data

- Metadata -> Relational Database, Graph Database

- Service for storing the images -> HTTP Server, PACS
Typical Machine Learning (ML) Workload

Visual Workload: Metadata + Visual Data

- Metadata -> Relational Database, Graph Database
- Service for storing the images -> HTTP Server, PACS
- Library for preprocessing -> OpenCV
Typical Machine Learning (ML) Workload

Visual Workload: Metadata + Visual Data

Metadata -> Relational Database, Graph Database

Service for storing the images -> HTTP Server, PACS

Library for preprocessing -> OpenCV
Typical Machine Learning (ML) Workload

Visual Workload: Metadata + Visual Data

Metadata -> Relational Database, Graph Database

Service for storing the images -> HTTP Server, PACS

Library for preprocessing -> OpenCV

Very tailored set of scripts
ML requires a new type of data management

- Unified
- Easy to use
- AI ready
- Efficient and evolving
Next Generation of Data Challenges

Primarily Visual
e.g. images, feature vectors, videos

Large scale
Rich with information
Individually large
Frequently time sensitive
Next Generation of Data Challenges

Primarily Visual
e.g. images, feature vectors, videos

Large scale
Rich with information
Individually large
Frequently time sensitive

Machine learning or data science usages

Expect easy integration
Repetitive data preprocessing
Ever evolving
Frequent network transfers
Next Generation of Data Challenges

Primarily Visual
  e.g. images, feature vectors, videos

Large scale
Rich with information
Individually large
Frequently time sensitive

VDMS is designed to address these

Machine learning or data science usages

Expect easy integration
Repetitive data preprocessing
Ever evolving
Frequent network transfers
VDMS Capabilities

- Efficient completion of complex metadata queries
  - Metadata stored in (persistent) memory
  - Using our in-house Graph Database (now ACID compliant)

- Efficient visual data retrieval
  - Images can be stored in image format designed for analytics
    - Threshold, crop, resize, or basic augmentation on images on the server side.
  - Visual Descriptors can be stored, and similarity search (KNN) performed on the fly.
    - Using different mechanism to index and compute distances
  - Video can be stored/retrieved.

- Straightforward client API to enable both metadata and data retrieval
  - Queries submitted as JSON (using Python or C++)
VDMS Pipeline Example

Python/C++ Client

ML pipeline

VDMS Client Module

VDMS Server

Metadata

Pre-processor

Data (images, videos, descriptors, blobs)

[1.52, 7.4, 0.87, 1]

[1.52, 7.4, 0.87, 1]

[1.52, 7.4, 98.7, ...]
VDMS Pipeline Example

Python/C++ Client

VDMS query

VDMS Client Module

JSON Query - Pull Data

Return Data

VDMS Server

Metadata

Data (images, videos, descriptors, blobs)

ML pipeline
VDMS Pipeline Example

Python/C++ Client

VDMS query

ML pipeline

VDMS Query + Image Blob

VDMS Client Module

JSON Query - Pull Data

Return Data

JSON Query - Push Data

Return Successful

VDMS Server

Metadata

Data (images, videos, descriptors, blobs)

Pre-processor

[1.52, 7.4, 98.7]

[1.52, 7.4, 98.7]

[1.52, 7.4, 98.7, ...]
VDMS
Release 2.0

- Properties on relationships
- Search using constraints on relationships
- Complex, multi-hop searches

- Feature vectors
- Videos
- Bounding boxes
- More visual data types besides Image

- PMGD 2.0
- VCL 1.0
- Wiki and docker updates
- Fixes, performance and usability updates

- Rotate, Flip
- Similarity search
- Richer set of operations
Visual Descriptors in VDMS

Novel solution for persistent Feature Vector storage, indexing, and search
Visual Descriptors in VDMS

Descriptor: [2.2, 2.9, 54.9, ...]
Label: person_28

Novel solution for persistent Feature Vector storage, indexing, and search
Visual Descriptors in VDMS

1. Descriptor: [2.2, 2.9, 54.9, ... ]
   Label: person_28

2. Query:
   Descriptor: [2.13, 3.3, 55.3, ... ]

Novel solution for persistent Feature Vector storage, indexing, and search
Visual Descriptors in VDMS

1. Descriptor: [2.2, 2.9, 54.9, ...]  
   Label: person_28

2.  

3. Query:  
   Descriptor: [2.13, 3.3, 55.3, ...]

4.  

5.  

[person_28, 4.2]  
[person_342, 654.32]  
[person_15, 786.43]  
[person_94, 965.65]

VDMS

Novel solution for persistent Feature Vector storage, indexing, and search
Visual ML Pipeline

Collaboration with Intel Labs
Integration with other Research Project

Data Acquisition
- Cameras
- Sensors

Edge Processing
- Power/compute limited
- Preprocessing
- Filtering
- Aggregation

SAF
- Processing data in real time

VDMS
- Persistence
- Intelligent access

Scanner
- Large sets of historical data

Presentation and Interpretation
More Information

- https://aperturedata.io
- https://github.com/IntelLabs/vdms

- **VDMS: Efficient Big-Visual-Data Access for Machine Learning Workloads**
  Luis Remis, Vishakha Gupta-Cledat, et. Al.
  Systems for Machine Learning Workshop @ NIPS 2018

- **Addressing the dark side of vision research: Storage**
  Vishakha Gupta-Cledat, Luis Remis, et al.
  ATC HotStorage 2017
The Snowflake Engine

Northwest Database Society (NWDS)
Annual Meeting 2019

Torsten Grabs - Product Management - torsten.grabs@snowflake.com
Who we are

Founded: August 2012
Mission: The data warehouse for the cloud

HQ in downtown San Mateo (south of San Francisco) with engineering offices in Bellevue, WA, and Berlin, Germany

1000+ employees, ~150 engs (and hiring...)
  - Founders: Benoit Dageville, Thierry Cruanes, Marcin Zukowski
  - CEO: Bob Muglia

GA in 2015
Raised over $900M across series A-F
Our Product

The Snowflake Elastic Data Warehouse, or “Snowflake”
- Multi-tenant, transactional, secure, highly scalable, elastic
- Implemented from scratch (no Hadoop, Postgres etc.)

Currently runs in the Amazon cloud (AWS) and Microsoft Azure
Serves millions of queries per day over 10s of petabyte of data
1500+ active customers, growing fast
Our Vision for a Cloud Data Warehouse

Data warehouse as a service
No infrastructure to manage, no knobs to tune

Multidimensional elasticity
On-demand scalability data, queries, users

All business data
Native support for relational + semi-structured data
Multi-cluster Shared-data Architecture

Cloud Services
- Infrastructure manager
- Optimizer
- Transaction Manager
- Security

Authentication & access control

Metadata

Virtual Warehouse
- Cache

Virtual Warehouse
- Cache

Virtual Warehouse
- Cache

Virtual Warehouse
- Cache

Data Storage

Rest (JDBC/ODBC/Python)

All data in one place

Independently scale storage and compute

No unload / reload to shut off compute

Every virtual warehouse can access all data
Data Storage Layer

Stores table data and query results

Uses cloud-based blob storage in AWS or Azure
  • Object store (key-value) with HTTP(S) PUT/GET/DELETE interface
  • High availability, extreme durability (11-9)

Some important differences w.r.t. local disks
  • Performance (sure...)
  • No update-in-place, objects must be written in full
  • But: can read parts (byte ranges) of objects

Strong influence on table file format and concurrency control
Table Files

Snowflake uses PAX [Ailamaki01] aka hybrid columnar storage for table files.

Tables horizontally partitioned into large immutable files (~16 MB each)

- Updates add or remove entire files
- Values of each column grouped together and compressed
- Queries read header + columns they need
- Old table versions retained for time travel

Metadata stored in a transactional key-value store (not blob storage)

- Which table consists of which blob storage objects
- Optimizer statistics, lock tables, transaction logs etc.
- Part of Cloud Services layer (see later)
Virtual Warehouse

VW = Cluster of cloud compute VM instances called worker nodes

Pure compute resources
- Created, destroyed, resized on demand
- Users may run multiple VW at same time
- Each VW has access to all data but isolated performance
- Users may shut down all VWs when they have nothing to run

T-Shirt sizes: XS to 4XL
- Users do not know which type or how many VM instances
- Service and pricing can evolve independent of cloud platform

Each worker node maintains local table cache
- Collection of table files i.e. cloud storage objects accessed in past
- Shared across concurrent and subsequent worker processes
- Assignment of table files to nodes using consistent hashing
Execution Engine

Columnar [MonetDB, C-Store, many more]
- Effective use of CPU caches, SIMD instructions, and compression

Vectorized [Zukowski05]
- Operators handle batches of a few thousand rows in columnar format
- Avoids materialization of intermediate results

Push-based [Neumann11]
- Operators push results to downstream operators (no Volcano iterators)
- Removes control logic from tight loops
- Works well with DAG-shaped plans

No transaction management, no buffer pool
- But: most operators (join, group by, sort) can spill to disk and recurse

© 2018 Snowflake Computing Inc. All Rights Reserved
Cloud Services

Collection of services
- Access control, query optimizer, transaction manager etc.

Heavily multi-tenant (shared among users) and always on
- Improves utilization and reduces administration

Each service replicated for availability and scalability
- Hard state stored in transactional key-value store
Concurrency Control

Designed for analytic workloads
  • Large reads, bulk or trickle inserts, bulk updates

Snapshot Isolation (SI) [Berenson95]

SI based on multi-version concurrency control (MVCC)
  • DML statements (insert, update, delete, merge) produce new table versions of tables by adding or removing whole files
  • Natural choice because table files in cloud storage are immutable
  • Additions and removals tracked in metadata (key-value store)

Versioned snapshots used also for time travel and cloning
Pruning

Database adage: The fastest way to process data? Don’t.
  • Limiting access only to relevant data is key aspect of query processing

Traditional solution: B⁺-trees and other indices
  • Poor fit for us: random accesses, high load time, manual tuning

Snowflake approach: pruning
  • AKA small materialized aggregates [Moerkotte98], zone maps [Netezza], data skipping [IBM]
  • Per file min/max values, #distinct values, #nulls, bloom filters etc.
  • Use metadata to decide which files are relevant for a given query
  • Smaller than indices, more load-friendly, no user input required
Ongoing Challenges

Scale
- Support thousands of concurrent users, some of which do *weird* things
- Metadata layer is becoming huge
- Customer data is becoming huge
- More cloud regions across the globe
- Categorizing and handling failures automatically is very hard
- *Automation* is key to keeping operations lean

Serverless computing paradigm

Continuous and low latency data ingestion

Data sharing and collaboration over data

Lots of other work left to do
- SQL performance improvements.
- Stronger integration with 3rd party tools
- Self-service model
- Multi-account manageability
- Data visualization

© 2018 Snowflake Computing Inc. All Rights Reserved
It’s a wrap

Snowflake is an enterprise-ready data warehouse as a service

- Novel multi-cluster, shared-data architecture
- Highly elastic and available
- Semi-structured and schema-less data at the speed of relational data
- Pure SaaS experience

Rapidly growing user base and data volume

Lots of challenging work left to do
Veritas: Overlaying Distributed Database Applications over Blockchains

Donald Kossmann
Microsoft Research
Value Prop of Blockchain = Proof for Digital Transactions
Transactions in the Real World

- All Transactions require Proof: Witnesses and/or Receipts
  - getting married (best man + ring)
  - buying a house (notary + contract)
  - drinking alcohol (driver's licence)

- Why do we need Proof?
  - transactions have conditions and come with rights & accountabilities
    - getting married: „I am married to you! Please, be nice to me!”
    - buying a house: „I am the rightful owner of the house! I am allowed to live here.“
    - birth: „I become Donald Kossmann.“ - drinking: „I am Donald Kossmann!“
  - Witnesses and receipts provide proof: Proof = Trust
Transactions in the Digital World

**PC Era:** If you are alone, you do not need trust (proof)
- user owns and controls all data; user trusts herself

**Cloud Era / Connected World:** Trust is needed
- users collaborate and share data (e.g., for AI)
- news gets hacked
- users need to verify data before making decisions: How?
  - proof (receipts & witnesses) in the digital world!
  - (Or we are stuck with trusted brands such as Facebook, ...)

4
Where is the Trust Button?

<table>
<thead>
<tr>
<th></th>
<th>First</th>
<th>Last</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tom</td>
<td>Cruise</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>Donald</td>
<td>Kossmann</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>Britney</td>
<td>Spears</td>
<td>35</td>
</tr>
</tbody>
</table>
Where is the Trust Button?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>First</td>
<td>Last</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Tom</td>
<td>Cruise</td>
<td>55</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Donald</td>
<td>Kossmann</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Britney</td>
<td>Spears</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Dream: Automate Trust

Create Proof in the Connected Digital World

- trace data and transactions
- every document comes with proof (verification)
- I can prove that I did the right thing

Blockchain is a nice building block, but not enough

- Issues: Integration, Performance, Privacy, ...
- Challenge: Retrofit trust into existing applications
Where is the Trust Button?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>First</td>
<td>Last</td>
<td>Age</td>
</tr>
<tr>
<td>2</td>
<td>Tom</td>
<td>Cruise</td>
<td>55</td>
</tr>
<tr>
<td>3</td>
<td>Donald</td>
<td>Kossmann</td>
<td>28</td>
</tr>
<tr>
<td>4</td>
<td>Britney</td>
<td>Spears</td>
<td>35</td>
</tr>
</tbody>
</table>
The Dream: Automate Trust

Create Proof in the Connected Digital World
- trace data and transactions
- every document comes with proof (verification)
- I can prove that I did the right thing

Blockchain is a nice building block, but not enough
- Issues: Integration, Performance, Privacy, ...
- Challenge: Retrofit trust into existing applications
Overview

Blockchain 101

Veritas: Integrating Proof into Databases

Example: Decentralized ID
Blockchain 101

**Idea 1:** Crypto to make transactions **immutable** and **atomic**.

**Idea 2:** *Consensus protocol* to commit a transaction. **Community** verifies all transactions.

\[
\text{Immutable Contract} \quad + \quad \text{Witnesses} \quad = \quad \text{Trust}
\]
BLOCKCHAIN 101: UNTRUSTED LEDGER

B1

<table>
<thead>
<tr>
<th>Amy</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harry</td>
<td>50</td>
</tr>
</tbody>
</table>

B2

20
Amy → Harry

B3

30
Amy → Harry

Ledger (linked list)
Problem: No protection against greedy Harry!
BLOCKCHAIN 101

- Padmasree: H(B3)
- Satya: H(B3)
- Teri: H(B3)

B1
- Amy: 70
- Harry: 50

B2
- H(B1)
- Amy → Harry: 50

B3
- H(B2)
- Amy → Harry: 30

Ledger (linked list)
TRADITIONAL VS. BLOCKCHAIN

Traditional IT Systems
- Productivity: great abstractions
- Security: proven technology
- Performance: millions ops/sec
- Standardization
- but no proofs

Blockchain
- reinvent the wheel
- but proofs
Veritas

- Separate Generation of Contracts (Idea 1) from Verification (Idea 2)
- Digital Register generates receipt, embedded in application
  - Blockchain service collects receipts and verifies log, separate from application
- Trust (world of Blockchains without Nudging Applications)
Example: Decentralized IDs

Acknowledgments:

Daniel Buchner, Esha Ghosh, Rahee Ghosh, Ankur Jain, Srinath Setty, Henry Tsai
"On the Internet, nobody knows you're a dog."

DID: Decentralized (Digital) Identifiers
Transactions in the Real World

- All Transactions require Proof: Witnesses and/or Receipts
  - getting married (best man + ring)
  - buying a house (notary + contract)
  - drinking alcohol (driver's licence)

- Why do we need Proof?
  - transactions have conditions and come with rights & accountabilities
    - getting married: „I am married to you! Please, be nice to me!“
    - buying a house: „I am the rightful owner of the house! I am allowed to live here.“
    - birth: „I become Donald Kossmann.“ - drinking: „I am Donald Kossmann!“
  - Witnesses and receipts provide proof: $\text{Proof} = \text{Trust}$
Identity Problem

Claims and Credentials
- Enter the country
  - Citizenship
- Drive a vehicle
  - Driving skills
- Drink alcohol
  - Age
- Enter this building
  - Work @ MS
Identity Systems Today

LinkedIn

SSN

Diplomas
Birth certificate
...

Facebook

Driver’s License

Google

MS Badge

Microsoft

Passport

US Bank

Me
Identity Systems Today: Challenges

• Lack of ownership and control over identifiers
  • Centralized root of trust

• Patchwork of multiple identifiers
  • Management complexity
  • Integration complexity (e.g., Mint)

• Non-cryptographic “proofs” of claims
  • Identity theft

• Privacy
  • Example: Establishing my age with DL reveals my location
Decentralized Identifiers (DIDs)

A self-owned identity which can be used to securely and privately store all elements of our identity and establish claims and credentials.
Decentralized Identity Foundation (DIF)
Decentralized Identifiers (DIDs)

A self-owned identity which can be used to securely and privately store all elements of our identity and establish claims and credentials.
Sidetree DID protocol
Sidetree DID protocol
Sidetree DID protocol

Create DID

did:ex:0f12d981115702
Sidetree DID protocol

Create DID

did:ex:0f12d981115702

```json
{
    "@context": "https://w3id.org/did/v1",
    "id": "did:ex:0f12d981115702",
    "authentication": [{
      // this key can be used to authenticate as did:...702
      "id": "did:example:123456789abcdefgfakekeys-1",
      "type": "RsaVerificationKey2018",
      "controller": "did:ex:0f12d981115702",
      "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\n"
    }], ...
}
```
DID-based Claims
DID-based Claims

Update

did:ex:0f12d98i115702
Name: Alice
DOB: 1/1/77

did:ex:0f12d98i115702
Name: Alice
DOB: 1/1/77
DID-based Claims

Alice

did:ex:0f12d98i115702

WA DOL

did:ex:1a996014456123

did:ex:1245a0b9d52328
DID-based claims: Proofs

 esk_Alice
 Alice

 Bob the Barman
DID-based claims: Proofs

$sk_{Alice}$

Alice

did:ex:0f12d98i115702

Bob the Barman
DID-based claims: Proofs

$sk_{Alice}$

Alice

$\text{did:ex:0f12d98i115702}$

Bob the Barman
DID-based claims: Proofs

$sk_{Alice}$

Alice

![Driver License Image]

Bob the Barman

did:ex:0f12d98i115702

```json
{
  "@context": "https://w3id.org/did/v1",
  "id": "did:ex:0f12d98i115702",
  "authentication": [{
    "id": "did:example:123456789abcdefg1#keys-1",
    "type": "RsaVerificationKey2018",
    "controller": "did:ex:0f12d98i115702",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY------\r\n"
  }],
  Name: Alice
  DL: 0ab346976fce34
}
```
DID-based claims: Proofs

$sk_{Alice}$

Alice

did:ex:0f12d98i115702

Bob the Barman

$pk_{Alice}$

SHA(DL$_{Alice}$)
DID-based claims: Proofs

\[ sk_{Alice} \]

Alice

Random string \( s \)

Bob the Barman

\[ pk_{Alice} \]

\[ SHA(DL_{Alice}) \]
DID-based claims: Proofs

$sk_{Alice}$

Random string $s$

$Sign(s, sk_{Alice})$

Bob the Barman

$pk_{Alice}$

SHA($DL_{Alice}$)
DID-based claims: Proofs

Alice

Bob the Barman

$sk_{Alice}$

$pk_{Alice}$

SHA($DL_{Alice}$)
DID-based claims: Proofs

$sk_{Alice}$

Alice

$DL_{Alice}$

Bob the Barman

$pk_{Alice}$

SHA($DL_{Alice}$)
Retrofitting OpenId

Relying Party

Alice

ID Provider

Claims Credentials
Retrofitting OpenId

Relying Party

Alice

Authentication

Token

ID Provider

Claims Credentials
Retrofitting OpenId

Relying Party

Account Setup
new DID

ID Provider

Alice

DID

Claims Credentials

DID
Retrofitting OpenId

[Diagram showing the interaction between a Relying Party, an ID Provider, and Alice. The diagram illustrates the flow of claims and credentials.]
Retrofitting OpenId

Relying Party

authenticate

ID Provider

Alice

Verifiable Caches

DID

Verifiable Caches

DID
Conclusions

- Blockchains
  - proofs of digital transactions
  - Limitations – abstractions, performance

- Veritas
  - Retrofit verifiability to existing systems
  - Overlay on blockchains for consensus

- Decentralized Ids
Retrofitting OpenId
Retrofitting OpenId
DID-based claims: Proofs

$sk_{Alice}$

Alice

Random string $s$

Bob the Barman

$pk_{Alice}$

SHA($DL_{Alice}$)