The Mimicking Octopus
Towards a one-size-fits-all Architecture for Database Systems

Alekh Jindal
Supervisor: Prof. Dr. Jens Dittrich

VLDB PhD Workshop
September 13, 2010
Database Landscape

Motivation

OLTP

Streaming System

OLAP

Archival System

Search Engine
Motivation

OLTP

OLAP

Archival System

Search Engine

Streaming System
Airline Company

Several Applications
Evolving Applications

Eventual Integration
ETL style data pipelines

Integration Cost
Licensing Cost
Maintenance Cost
DBA Cost
Engineering Cost

Motivation

OLTP
OLAP
Streaming
Search
Archival

Cheap Fares
Ticket Booking
Flight Search
Booking Archives
Reporting

Hard-coded optimizations
Hard-coded data layouts

OLTP
OLAP
Problem Statement

• Single database system
• Automatic adaption
• Improved performance
• Lower cost
• Better maintainability
OctopusDB Overview

• One-size-fits-all architecture
• Abstract storage concept: Storage Views (SV)
• Single optimization problem: SV Selection
• Holistic SV optimizer
System Architecture

- No hard-coded store
- All operations recorded as logical log entries in a primary log on stable storage using WAL

OctopusDB
System Architecture

- No hard-coded store
- All operations recorded as logical log entries in a primary log on stable storage using WAL
System Architecture

- No hard-coded store
- All operations recorded as logical log entries in a primary log on stable storage using WAL
Storage Views

- Arbitrary physical representations of data
- Different layouts under a single umbrella
Storage Views

• Arbitrary physical representations of data
• Different layouts under a single umbrella

Primary
Log SV
Row SV
Column SV
Index SV
Storage Views

• Arbitrary physical representations of data
• Different layouts under a single umbrella

<table>
<thead>
<tr>
<th>Primary</th>
<th>Secondary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log SV</td>
<td>Partial Index SV</td>
</tr>
<tr>
<td>Row SV</td>
<td>Bag-partitioned SV</td>
</tr>
<tr>
<td>Column SV</td>
<td>Key-consolidated SV</td>
</tr>
<tr>
<td>Index SV</td>
<td>Vertically/Horizontally Partitioned SV</td>
</tr>
</tbody>
</table>
### Storage Views

- Arbitrary physical representations of data
- Different layouts under a single umbrella

<table>
<thead>
<tr>
<th>Primary</th>
<th>Secondary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log SV</td>
<td>Partial Index SV</td>
</tr>
<tr>
<td>Row SV</td>
<td>Bag-partitioned SV</td>
</tr>
<tr>
<td>Column SV</td>
<td>Key-consolidated SV</td>
</tr>
<tr>
<td>Index SV</td>
<td>Vertically/Horizontally Partitioned SV</td>
</tr>
</tbody>
</table>

...any hybrid combination of the above
Use-case Scenario*

- Flight booking system
- Tables: Tickets, Customers
- Tickets: several attributes, frequently updated
- Customers: fewer attributes

Queries:

```sql
SELECT C.*
FROM Tickets T, Customers C
WHERE T.customer_id=C.id AND T.a1=x1 AND T.a2=x2 ... AND T.an=xn
```

Flight Booking System

SELECT C.*
FROM Tickets T, Customers C
WHERE T.customer_id = C.id
AND T.a1 = x1 .... AND T.an = xn
Bag-partitioning

SELECT C.* 
FROM Tickets T, Customers C 
WHERE T.customer_id=C.id 
AND T.a1=x1 .... AND T.an=xn

customers, 01, <tom, 25, 
customers, 02, <marc, 23, 
customers 03, <felix, 20, 
customers, 03, <felix, 20, 

tickets, 301, <paris, rome, E,...> 
tickets, 302, <moscow, berlin, B,...> 
tickets, 303, <tokyo, beijing, E,...> 
tickets, 303, <tokyo, beijing, B,..> 

.....

.....

September 13, 2010  Towards a one-size-fits-all Database Architecture - Alekh Jindal
Key-consolidation

SELECT C.*
FROM Tickets T, Customers C
WHERE T.customer_id = C.id
AND T.a1 = x1 .... AND T.an = xn

[customers, 01, <tom, 25, ...]
customers, 02, <marc, 23, ...]
customers 03, <felix, 20, ...]
customers, 03, <felix, 20, ...]
tickets, 301, <paris, rome, E,...>
tickets, 302, <moscow, berlin, B,...>
tickets, 303, <tokyo, beijing, E,...>
tickets, 303, <tokyo, beijing, B,...>
......
tickets, 301, <paris, rome, E,...>
tickets, 302, <moscow, berlin, B,...>
tickets, 303, <tokyo, beijing, B,...>
tickets, 303, <tokyo, beijing, B,..>
......
tickets, 301, <paris, rome, E,...>
tickets, 302, <moscow, berlin, B,...>
tickets, 303, <tokyo, beijing, B,..>
......
customers, 01, <tom, 25, tom@abc.com, ...> customers, 02, <marc, 23, marc@abc.com, ...> customers, 03, <felix, 20, felix@xyz.com, ...>
......
Storage View Transformation

Customers

Tickets

Log SV

Col SV

Row SV

Result

SELECT C.*
FROM Tickets T, Customers C
WHERE T.customer_id = C.id
AND T.a1 = x1 ... AND T.an = xn

customers, 01, <tom, 25, ...>
customers, 02, <marc, 23, ...>
customers, 03, <felix, 20, ...>
customers, 03, <felix, 20, ...>

tickets, 301, <paris, rome, E,...>
tickets, 302, <moscow, berlin, B,...>
tickets, 303, <tokyo, beijing, E,...>
tickets, 303, <tokyo, beijing, B,...>

......

customers, 01, <tom, 25, ...>
customers, 02, <marc, 23, ...>
customers, 03, <felix, 20, ...>
customers, 03, <felix, 20, ...>

tickets, 301, <paris, rome, E,...>
tickets, 302, <moscow, berlin, B,...>
tickets, 303, <tokyo, beijing, B,...>
tickets, 303, <tokyo, beijing, B,...>

......

customers, 01, <tom, 25, tom@abc.com, ...>
customers, 02, <marc, 23, marc@abc.com, ...>
customers, 03, <felix, 20, felix@xyz.com, ...>
customers, 03, <felix, 20, felix@xyz.com, ...>

......

.....
Towards a one-size-fits-all Database Architecture - Alekh Jindal

Information Systems Group

Hot-Cold Storage Views

SELECT C.*
FROM Tickets T, Customers C
WHERE T.customer_id = C.id
AND Ta1=x1 AND Ta2=xn

Col SV

Row SV

Storage Views

Result

customers

tickets

Tickets

Customers

Customers

tickets

Recent bag, key

Bag View

Log SV
Index Storage Views

SELECT C.*
FROM Tickets T, Customers C
WHERE T.customer_id=C.id
AND T.a1=x1 .... AND T.an=xn

Towards a one-size-fits-all Database Architecture - Alekh Jindal
Isn’t this same as Materialized Views?
Index Storage Views

Isn't this same as Materialized Views?

NO!
Materialized View knows what to materialize
Index Storage Views

Storage View also knows **how** to materialize

SELECT C.*
FROM Tickets T, Customers C
WHERE T.customer_id=C.id
AND T.a1=x1 ...
AND T.an=xn
Index Storage Views

Storage View also knows **how** to materialize

A Materialized View still needs a Storage View

**SELECT C.* FROM Tickets T, Customers C WHERE T.customer_id=C.id AND T.a1=x1 AND T.an=xn**
Storage View Selection

SELECT C.*
FROM Tickets T, Customers C
WHERE T.customer_id=C.id
AND T.a1=x1 AND T.an=xn
Storage View Selection

Pick right Storage Views to:
create, update, query and drop

SELECT C.*
FROM Tickets T, Customers C
WHERE T.customer_id=C.id
AND T.a1=x1 .... AND T.an=xn
Storage View Selection

Single Optimization Problem: “Storage View Selection”

SELECT C.*
FROM Tickets T, Customers C
WHERE T.customer_id = C.id
AND T.a1 = x1 .... AND T.an = xn
Holistic Storage View Optimizer

- Storage totally dynamic: *Any* subset of data in *Any* storage structure
- Storage View selection
- Storage View update maintenance
- Pick physical execution plan
- Combine results spanning several Storage Views
Research Challenges

- Single umbrella for different storage layouts
  - storage layer abstraction
  - still layout specific specialization

- Automatic adaptive bifurcation
  - monolithic system
  - right online algorithms

- Simplicity vs Optimization
  - only as complex as required
  - mimic several specialized systems
Related Work

- Materialized Views [Chirkova et. al. VLDBJ 2002]
  - as pointed before different from storage views

- Dynamic materialized views [Zhou et. al. ICDE 2007]
  - horizontal dynamism, storage view still open

- View matching, query containment [A.Y. Halevy VLDBJ 2001]
  - again operate on a higher level

- Cracked databases [Idreos et. al. CIDR 2007]
  - logical partitioning of data, only horizontal

- Rodent store [Cudre-Mauroux et. al. CIDR 2009]
  - still assumes a store

- GMAP [Tsatalos et. al. VLDB 1994]
  - does not adapt the stores
Optimizer Cost Model

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{\text{log scan}}(N)$</td>
<td>Log SV scan cost</td>
<td>$\frac{\sum_{i=1}^{N} \text{colsize(log)}<em>i}{m} \cdot C</em>{\text{random}} + \frac{\sum_{i=1}^{N} \text{colsize(log)}_i}{\text{pageSize}} / BW$</td>
</tr>
<tr>
<td>$C_{\text{row scan}}(N)$</td>
<td>Row SV scan cost</td>
<td>$\frac{N \cdot \sum_{A_i \in A} \text{colsize}(A_i)}{m} \cdot C_{\text{random}} + \frac{N \cdot \sum_{A_i \in A} \text{colsize}(A_i)}{\text{pageSize}} / BW$</td>
</tr>
<tr>
<td>$C_{\text{col scan}}(N, S)$</td>
<td>Col SV scan cost</td>
<td>$\sum_{A_i \in S} \left( \frac{\sum_{A_i \in S} \text{colsize}(A_i)}{m} \cdot C_{\text{random}} + \frac{N \cdot \text{colsize}(A_i)}{\text{pageSize}} \right) / BW$</td>
</tr>
<tr>
<td>$C_{\text{index lookup}}(N)$</td>
<td>Index lookup cost</td>
<td>$C_{\text{random}} \cdot \left( \log P(N \cdot (\text{colsize(key)} + \text{pointerSize}) / \text{pageSize} \right)$</td>
</tr>
<tr>
<td>$C_{\text{row cl. index scan}}(N, sel)$</td>
<td>Unclustered Indexed Row SV scan cost</td>
<td>$C_{\text{lookup}}(N) + C_{\text{scan}}([sel \cdot N])$</td>
</tr>
<tr>
<td>$C_{\text{col cl. index scan}}(N, S, sel)$</td>
<td>Unclustered Indexed Col SV scan cost</td>
<td>$C_{\text{lookup}}(N) + C_{\text{col scan}}([sel \cdot N], S)$</td>
</tr>
<tr>
<td>$C_{\text{row uncl. index scan}}(N, sel)$</td>
<td>Clustered Indexed Row SV scan cost</td>
<td>$C_{\text{index lookup}} + [sel \cdot N] \cdot (C_{\text{random}} + \text{pageSize/BW})$</td>
</tr>
<tr>
<td>$C_{\text{col uncl. index scan}}(N, S, sel)$</td>
<td>Unclustered Indexed Col SV scan cost</td>
<td>$C_{\text{index lookup}} + [sel \cdot N] \cdot</td>
</tr>
</tbody>
</table>

### Query Cost Model

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{\text{log update}}(N_u)$</td>
<td>Log SV update cost</td>
</tr>
<tr>
<td>$C_{\text{row update}}(N, N_u)$</td>
<td>Row SV update cost</td>
</tr>
<tr>
<td>$C_{\text{col update}}(N, N_u, S)$</td>
<td>Col SV update cost</td>
</tr>
<tr>
<td>$C_{\text{index split}}(d)$</td>
<td>Index split cost</td>
</tr>
<tr>
<td>$C_{\text{row cl. index update}}(N, N_u, d)$</td>
<td>Cl. Index Row SV update cost</td>
</tr>
<tr>
<td>$C_{\text{col cl. index update}}(N, N_u, S, d)$</td>
<td>Cl. Index Col SV update cost</td>
</tr>
<tr>
<td>$C_{\text{row uncl. index update}}(N, N_u, d)$</td>
<td>Uncl. Index Row SV update cost</td>
</tr>
<tr>
<td>$C_{\text{col uncl. index update}}(N, N_u, S, d)$</td>
<td>Uncl. Index Col SV update cost</td>
</tr>
</tbody>
</table>

### Transform Cost Model

<table>
<thead>
<tr>
<th>SV Transformation</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log SV $\rightarrow$ Row SV</td>
<td>$C_{\text{log scan}}(N) + C_{\text{row scan}}(N)$</td>
</tr>
<tr>
<td>Log SV $\rightarrow$ Col SV</td>
<td>$C_{\text{log scan}}(N) + C_{\text{col scan}}(N, A)$</td>
</tr>
<tr>
<td>Row SV $\rightarrow$ Col SV</td>
<td>$C_{\text{row scan}}(N) + C_{\text{col scan}}(N, A)$</td>
</tr>
<tr>
<td>Row SV $\leftrightarrow$ Col SV</td>
<td>$C_{\text{row scan}}(N) + \left( \frac{F+1}{F-1} \right) \cdot C_{\text{random}}$</td>
</tr>
<tr>
<td>Col SV $\rightarrow$ Index SV</td>
<td>$C_{\text{col scan}}(N, {\text{key, rowID}}) + \left( \frac{F+1}{F-1} \right) \cdot C_{\text{random}}$</td>
</tr>
</tbody>
</table>

### Further Directions

- Towards a one-size-fits-all Database Architecture - Alekh Jindal

September 13, 2010
Comparing Different Stores

<table>
<thead>
<tr>
<th>Tuples</th>
<th>Tickets</th>
<th>Customers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selectivity</td>
<td>0.9</td>
<td>0.1</td>
</tr>
<tr>
<td>Attributes Referenced</td>
<td>4/20</td>
<td>20/20</td>
</tr>
</tbody>
</table>

![Comparison of Different Stores](chart)

- **Row Store**
- **Column Store**
- **Indexed Row Store**
- **Indexed Column Store**
- **Fractured Mirrors**
- **Indexed Fractured Mirrors**

**Query Costs**

**Update Costs**

**Workload time [seconds]**

- **Tickets**: 100,000
- **Customers**: 20,000

**Attributes Referenced**

- **Referenced**: 4/20
- **20/20**
Next Steps

1. Automatically picking the right layout  
   - row, column, partitioned, cracked, more?

2. Storage View compression  
   - adaptive compression

3. Storage View maintenance  
   - maintaining heterogenous SVs

4. OctopusDB benchmarking and evaluation  
   - one-size-fits-all benchmark
**Summary**

**Airline Company**

- Several Applications
- Evolving Applications
- Eventual Integration
- ETL style data pipelines
- Hard-coded optimizations
- Hard-coded data layouts

**Database Landscape**

- OLTP
- OLAP
- Streaming System
- Archival System
- Search Engine

**Licensing Cost**

- DBA Cost
- Maintenance Cost
- Engineering Cost
- Integration Cost

**Motivation**

- Hard-coded optimizations

**OLTP**

- Booking
- Tickets
- Archives

**OLAP**

- Reporting
- Analytics
- Search

**Streaming**

- Real-time data
- Event processing

**Archival**

- Historical data
- Archiving

**Search Engine**

-全文检索
- 面向客户查询

**System Components**

- API
- Storage View Store
- Primary Log Store
- Log SV
- Recovery Manager
- Query Catalog
- Storage View Catalog
- Holistic SV Optimizer
- Transaction Manager

**Query Costs**

- 0.0
- 0.2
- 0.4
- 0.6
- 0.8
- 1.0

**Update Costs**

- Row Store
- Column Store
- Indexed Row Store
- Indexed Column Store
- Indexed Mirrors
- Fractured Mirrors
- OctopusDB

**Workload time [seconds]**

**SELECT C.*
FROM Tickets T, Customers C
WHERE T.customer_id=C.id
AND T.a1=x1 ... AND T.an=xn