Distributed Storage and Interactive Analytics for Graph-Structured Data

Alvin AuYoung
NSF CIFellow, HP Labs

Michael M. Lee*, Indrajit Roy, Vanish Talwar, YY Zhou*
HP Labs*, UC San Diego
Challenges for Graph Storage & Analytics

Scale
Query rate
Extensibility
Updates

100K+ third party apps query social graph
200K+ events/sec

62M active users, graph analysis for player retention

30K+ sensors in California, readings per minute
## State of the art

<table>
<thead>
<tr>
<th>Technology</th>
<th>Scalable</th>
<th>Queries</th>
<th>Extensible</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>No</td>
<td>Slow</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MySQL+cache</td>
<td>Yes</td>
<td>Fast, stale</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Batch frameworks</td>
<td>Yes</td>
<td>Fast, offline</td>
<td>Some</td>
<td>No</td>
</tr>
<tr>
<td>GraphDBs</td>
<td>No</td>
<td>Fast, online</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Overall - Limited scalability, performance, functionality**
# Design Goals

<table>
<thead>
<tr>
<th>Technology</th>
<th>Scalable</th>
<th>Queries</th>
<th>Extensible</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>No</td>
<td>Slow</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MySQL+cache</td>
<td>Yes</td>
<td>Fast, stale</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Batch frameworks</td>
<td>Yes</td>
<td>Fast, offline</td>
<td>Some</td>
<td>No</td>
</tr>
<tr>
<td>GraphDBs</td>
<td>No</td>
<td>Fast, online</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Concerto</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Our solution: Concerto

<table>
<thead>
<tr>
<th>Technology</th>
<th>Scalable</th>
<th>Queries</th>
<th>Extensible</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concerto</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Distributed Storage

- **Clients**: Concerto
  - CreateVertex(..)
  - AllocateSlab(..)

- **Graph allocators**

- **Global address space**

- **Servers**: Sinfonia memnodes
Our solution: Concerto

<table>
<thead>
<tr>
<th>Technology</th>
<th>Scalable</th>
<th>Queries</th>
<th>Extensible</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concerto</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Design Choice</td>
<td>Distributed storage</td>
<td></td>
<td></td>
<td>Distributed transactions</td>
</tr>
</tbody>
</table>

- Technology Scalable Extensible Updates
- Concerto Yes Yes Yes Yes
- Design Choice Distributed storage Distributed transactions
Graph Elements within one machine

CreateVertex(..)

Graph Allocator

Buddy memory allocator

Static size block

Global address space

Server (memnode)

Node object

Edge object

Property object (variable size)
Graph Elements within one machine

Memnode
Graph Elements within one machine

Static size block

FreePtr

Memnode
Graph Elements within one machine

- Michael

- Static size block

- M

- FreePtr

- Memnode
Graph Elements within one machine

Static size block

Memnode

FreePtr

Indrajit

Michael
Graph Elements within one machine

Static size block

Memnode

FreePtr

M I M I

Indrajit

Michael
Graph Elements within one machine

- Indrajit
- Michael
- Partha

Static size block

Memnode
FreePtr
Graph Elements within one machine

Indrajit

Michael

Partha

Static size block

Memnode

FreePtr
Graph Elements within one machine
Graph Elements within one machine
Graph Elements within one machine

Grad: UCSD  
Emp: HPIntern

Indrajit

Michael

Partha

Vanish

Memnode

M  I  M  I  P

I  P  M  P  V  M  V  P
Graph Elements within one machine

Grad: UCSD
Emp: HPIntern

Indrajit
Michael
Partha
Vanish

Memnode
Graph Elements within one machine

Grad: UCSD
Emp: HPIntern

Memnode

Key 'Grad'

Key 'Emp'
Graph Elements within one machine

Grad: UCSD
Emp: HPIntern

Indrajit
Partha
Vanish
Michael

Memnode
Graph Elements within one machine

Grad: UCSD
Emp: HPIntern

Indrajit
Michael
Partha
Vanish

Memnode

Key 'Grad'
Value 'UCSD'

Key 'Emp'
Value 'HPIntern'

M | I | M | I | P
Key 'Grad'
Value 'UCSD'

I | P | M | P | V | M | V | P
Our solution: Concerto

<table>
<thead>
<tr>
<th>Technology</th>
<th>Scalable</th>
<th>Queries</th>
<th>Extensible</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concerto</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Design Choice</td>
<td>Distributed storage</td>
<td>In-memory data structs</td>
<td>Distributed transactions</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:
- Indrajit
- Partha
- Vanish
- Han
- Stan
- Michael
- Zach
- Stefan
- John
Example graph query: k-hop

Average degree between 2 random users is 4.74
Illustration: Distributed k-hop traversal

Get 3-hop of Michael
Illustration: Distributed k-hop traversal
Illustration: Distributed k-hop traversal

Get 3-hop of Michael

Init & start

Exchange BFS state information

Get 1-hop

2hop operation

Partha

Indrajit

Vanish

Han

Michael

Stan

Zach

Stefan

John

Partha

Vanish

Eh B F S t t

Eh x c h a n g e B F S s t a t e i n f o r m a t i o n
Illustration: Distributed k-hop traversal

Get 3-hop of Michael

Init & start

Get 1-hop

2hop operation

3hop operation
Illustration: Distributed k-hop traversal

- Get 3-hop of Michael
  - 2hop operation
  - 3hop operation
  - Retrieve 3-hop results

- Get 1-hop
  - Init & start

Nodes:
- Indrajit
- Zach
- Stefan
- Han
- John
- Partha
- Vanish
- Michael
- Stan
## Our solution: Concerto

<table>
<thead>
<tr>
<th>Technology</th>
<th>Scalable</th>
<th>Queries</th>
<th>Extensible</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concerto</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Design Choice</td>
<td>Distributed storage</td>
<td>In-memory data structs</td>
<td>Distributed transactions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Parallel in-server processing</td>
<td></td>
</tr>
</tbody>
</table>
K-hop evaluation (Twitter: 51M v, 1B e)

Random source, avg of 3 runs, < 4% stdev/avg
K-hop evaluation (Twitter: 51M v, 1B e)

Random source, avg of 3 runs, < 4% stdev/avg
K-hop evaluation (Twitter: 51M v, 1B e)

Also, k-core algorithm: **26x faster** than Neo4J, MySQL

**Distributed k-core** improves from Concerto-1 to Concerto-64
## Our solution: Concerto

<table>
<thead>
<tr>
<th>Technology</th>
<th>Scalable</th>
<th>Queries</th>
<th>Extensible</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concerto</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
| Design Choice   | Distributed storage | In-memory data structs | Distributed transactions |-
|                 | Parallel in-server processing |-

Extensibility use case...
Incident Impact Analysis*

30K+ sensors in California, readings per minute

*[miller,gupta 2011] Stanford/HP Labs
Graph Views and Events

- Applications define **views**: sub-graphs of interest
- Applications register **event handlers** on views
Graph Views and Events

- Applications define **views**: sub-graphs of interest
- Applications register **event handlers** on views
## Our solution: Concerto

<table>
<thead>
<tr>
<th>Technology</th>
<th>Scalable</th>
<th>Queries</th>
<th>Extensible</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concerto</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Design Choice</td>
<td>Distributed storage</td>
<td>In-memory data structs</td>
<td>Views</td>
<td>Distributed transactions</td>
</tr>
<tr>
<td></td>
<td>Parallel in-server processing</td>
<td></td>
<td>Event-handlers</td>
<td></td>
</tr>
</tbody>
</table>

Evaluate extensibility use case...
Incident Impact Analysis (Road-CA: 2M v, 5M e)

Execution time (seconds)

- Concerto+EV Incident 1
- Concerto Incident 1
- Concerto+EV Incident 2
- Concerto Incident 2

- Compute
- Update
- Queuing
- Polling
Incident Impact Analysis (Road-CA: 2M v, 5M e)
## Summary

<table>
<thead>
<tr>
<th>Technology</th>
<th>Scalable</th>
<th>Queries</th>
<th>Extensible</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>No</td>
<td>Slow</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MySQL+cache</td>
<td>Yes</td>
<td>Fast, stale</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Batch frameworks</td>
<td>Yes</td>
<td>Fast, offline</td>
<td>Some</td>
<td>No</td>
</tr>
<tr>
<td>GraphDBs</td>
<td>No</td>
<td>Fast, online</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Concerto</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Design choice
- Distributed storage
- In-memory data structs
- Events and views
- Distributed transactions
- Parallel in-server processing

### Ongoing work
- Evaluate impact of existing graph partitioning techniques
- Evaluate more graph algorithms and real-world networks
Auxiliary Slides
## Evaluation workloads

<table>
<thead>
<tr>
<th>Graph</th>
<th>Vertices</th>
<th>Edges</th>
<th>File size</th>
<th>Experiments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter-S</td>
<td>33M</td>
<td>282M</td>
<td>6.5 GB</td>
<td>Insertion, k-hop</td>
</tr>
<tr>
<td>Twitter-L</td>
<td>51M</td>
<td>1B</td>
<td>38 GB</td>
<td></td>
</tr>
<tr>
<td>Social-S</td>
<td>3M</td>
<td>13M</td>
<td>197 MB</td>
<td>Insertion, k-core</td>
</tr>
<tr>
<td>Social-L</td>
<td>90M</td>
<td>405M</td>
<td>7.5 GB</td>
<td></td>
</tr>
<tr>
<td>Road-CA</td>
<td>2M</td>
<td>5M</td>
<td>84 MB</td>
<td>Traffic analysis</td>
</tr>
</tbody>
</table>
## Insertion throughput

<table>
<thead>
<tr>
<th>Inserts/sec</th>
<th>Vertices</th>
<th></th>
<th>Edges</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Single</td>
<td>Bulk</td>
<td>Single</td>
<td>Bulk</td>
</tr>
<tr>
<td>Neo4J</td>
<td>282</td>
<td>6,120</td>
<td>337</td>
<td>6,467</td>
</tr>
<tr>
<td>MySQL</td>
<td>15,076</td>
<td>119,422</td>
<td>13,788</td>
<td>61,446</td>
</tr>
<tr>
<td>Concerto-1</td>
<td>6,358</td>
<td>1,167,970</td>
<td>7,088</td>
<td>940,857</td>
</tr>
<tr>
<td>Concerto-10</td>
<td>29,695</td>
<td>2,625,011</td>
<td>27,122</td>
<td>1,873,476</td>
</tr>
</tbody>
</table>
K-hop evaluation (Twitter: 51M v, 1B e)

<table>
<thead>
<tr>
<th>Concerto</th>
<th>1-hop</th>
<th>2-hop</th>
<th>3-hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>121.5</td>
<td>623.64</td>
<td>682.28</td>
</tr>
<tr>
<td>Stdev</td>
<td>291.02</td>
<td>521.75</td>
<td>322.72</td>
</tr>
</tbody>
</table>

Normalized Throughput: $\frac{|k\text{-hop set}|}{\text{latency (ms)}}$