Gromit
An In-Memory Graph Database

Yunling Cui

University of Waterloo

yunling.cui@uwaterloo.ca

February 10, 2017

Overview

Social Network Applications

[3]

Relational Database Management System (RDBMS)

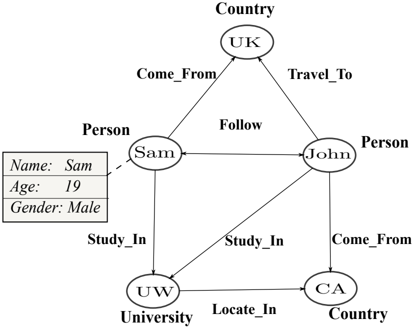


Table: Person			
P.ID	Name	U.ID	C.ID
0	Sam	2	4
1	John	2	3

Table: University			
U.ID	Name	Abbr.	C.ID
2	University of Waterloo	UW	3
5	University of Toronto	UofT	3

Table: Country		
C.ID	Name	Capital
3	Canada	Ottawa
4	UK	London

Relational Database Management System (RDBMS)

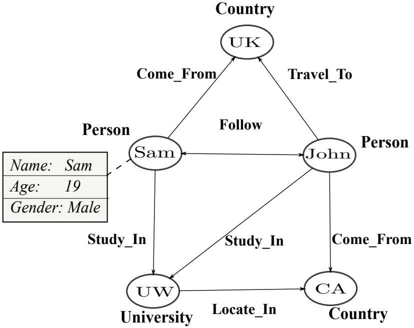


Table: Person			
P.ID	Name	U.ID	C.ID
0	Sam	2	4
1	John	2	3

Table: University			
U.ID	Name	Abbr.	C.ID
2	University of Waterloo	UW	3
5	University of Toronto	UofT	3

Table: Country		
C.ID	Name	Capital
3	Canada	Ottawa
4	UK	London

NoSQL Store

Examples

- Wide-Column Store [1]
- Key-Value Store [4]
- Document Store [2]

Person Document	
ID	0
Name	Sam
University	2
Country	4

Person Document	
ID	1
Name	John
University	2
Country	3

University Document	
ID	2
Name	University of Waterloo
Abbr.	UW
Country	3

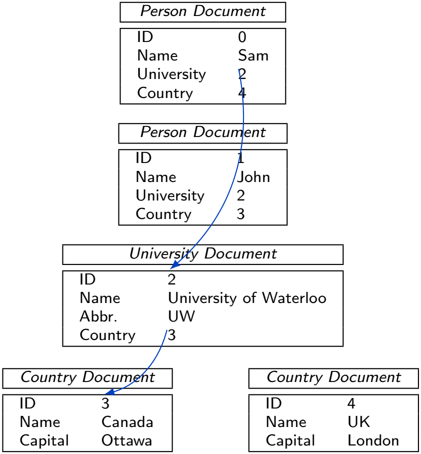
Country Document	
ID	3
Name	Canada
Capital	Ottawa

Country Document	
ID	4
Name	UK
Capital	London

NoSQL Store

Examples

- Wide-Column Store [1]
- Key-Value Store [4]
- Document Store [2]



Graph Store

Table: Person

P.ID	Name	U.ID	C.ID
0	Sam	2	4
1	John	2	3

Table: University

U.ID	Name	Abbr.	C.ID
2	University of Waterloo	UW	3
5	University of Toronto	UofT	3

Table: Country

C.ID	Name	Capital
3	Canada	Ottawa
4	UK	London

Person Document

ID	0
Name	Sam
University	2
Country	4

Person Document

ID	1
Name	John
University	2
Country	3

University Document

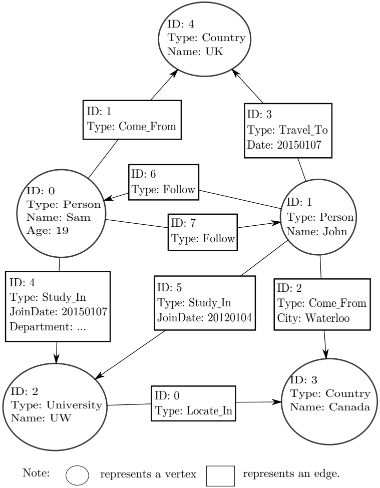
ID	2
Name	University of Waterloo
Abbr.	UW
Country	3

Country Document

ID	3
Name	Canada
Capital	Ottawa

Country Document

ID	4
Name	UK
Capital	London



Motivation

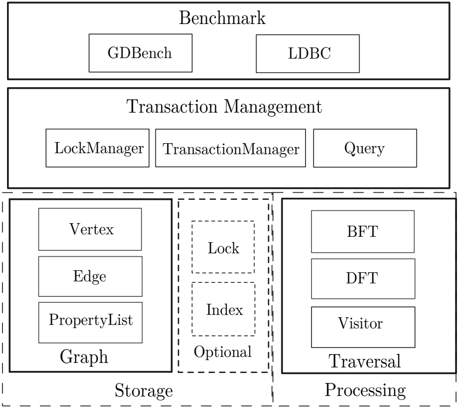
- Graph databases are suitable for highly connected data
- Simulators work well with applications written in C++

Motivation

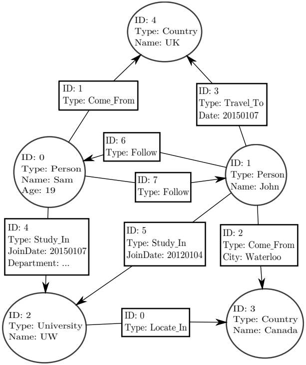
- Graph databases are suitable for highly connected data
- Simulators work well with applications written in C++

Objective: To build a graph database backend in C++

Main Contributions



Graph Storage



Vertex			
ID	*Label*	*PropertyList*	*NextEdge*
0	Person	Pv0	E7
1	Person	Pv1	E7
...			
4	Country	Pv4	E3
...			

PropertyList	
Key	*Value*
First Name	Sam
Last Name	Smith
...	
Date of Birth	...
...	

PropertyList	
Key	*Value*
Name	UK
Capital City	London
...	

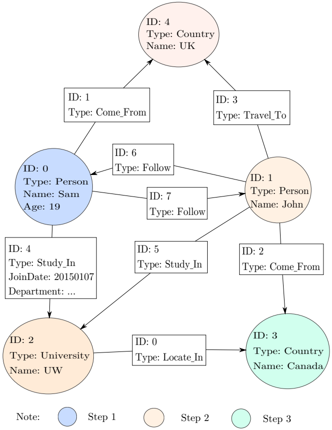
Edge								
ID	*Label*	*PropertyList*	*FirstVertex*	*SecondVertex*	*FNE*	*FPE*	*SNE*	*SPE*
...								
1	Follow	Pe1	V0	V4	E4	E7		E3
3	Travel_To	Pe1	V1	V4	E2	E7	E1	
...								
7	Follow	Pe1	V0	V1	E1		E3	
...								

Graph Processing

- Breadth-First Search (BFS)
- Depth-First Search (DFS)

Example

```
function BFS(Graph, Node)
    create empty set S
    create empty Queue Q
    Q.enqueue(Node)
    S.insert(Node)
    while !Q.empty() do
        Current = Q.dequeue()
        for all n in Current.neighbor do
            if !S.has(n) then
                S.insert(n)
                Q.enqueue(n)
            end if
        end for
    end while
end function
```



Query Framework with Visitor

```
function BFT(Graph, Node, Visitor)
    create empty set S
    create empty Queue Q
    Q.enqueue(Node)
    S.insert(Node)
    while !Q.empty() do
        Current = Q.dequeue()
        for all n in Current.neighbor do
            if !S.has(n) then
```

Query Framework with Visitor

```
function BFT(Graph, Node, Visitor)
    create empty set S
    create empty Queue Q
    Q.enqueue(Node)
    S.insert(Node)
    while !Q.empty() do
        Current = Q.dequeue()
        for all n in Current.neighbor do
            if !S.has(n) then
```

Filter 1	filterLabel("University"); filterDirection(OUT);
Filter 2	filterDepth(1)

Query Framework with Visitor

```
function BFT(Graph, Node, Visitor)
    create empty set S
    create empty Queue Q
    Q.enqueue(Node)
    S.insert(Node)
    while !Q.empty() do
        Current = Q.dequeue()
        for all n in Current.neighbor do
            if !S.has(n) then
                if Visitor.visitNext(n) then
                    return
                end if
                S.insert(n)
                Q.enqueue(n)
            end if
        end for
    end while
end function
```

Query Framework with Visitor

```
function BFT(Graph, Node, Visitor)
    create empty set S
    create empty Queue Q
    Q.enqueue(Node)
    S.insert(Node)
    while !Q.empty() do
        Current = Q.dequeue()
        for all n in Current.neighbor do
            if !S.has(n) then
                if Visitor.visitNext(n) then
                    return
                end if
                S.insert(n)
                Q.enqueue(n)
            end if
        end for
    end while
end function
```

- Selection
- Summarization
- Path searching
- Pattern matching
- Expression calculation

Transaction Management

Transaction Atomicity

- Transaction is a logical unit of such operations
- Each transaction finishes all operations or none

Transaction Management

Transaction Atomicity

- Transaction is a logical unit of such operations
- Each transaction finishes all operations or none

Locking Mechanisms

- Concurrency control mechanisms are required to protect data
- Two-Phase locking is implemented in Gromit

Transaction Management

Transaction Atomicity

- Transaction is a logical unit of such operations
- Each transaction finishes all operations or none

Locking Mechanisms

- Concurrency control mechanisms are required to protect data
- Two-Phase locking is implemented in Gromit

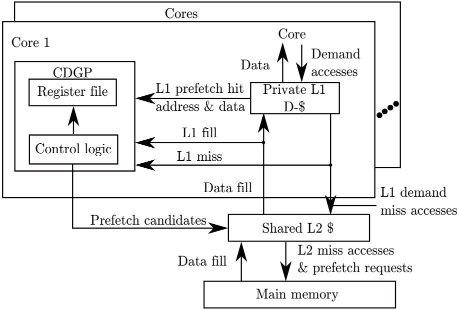
Deadlock Prevention Techniques

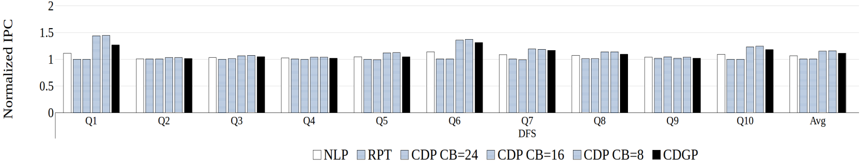
- No-Wait [5]
- Wait-Die
- Deadlock Detection

Benchmarks

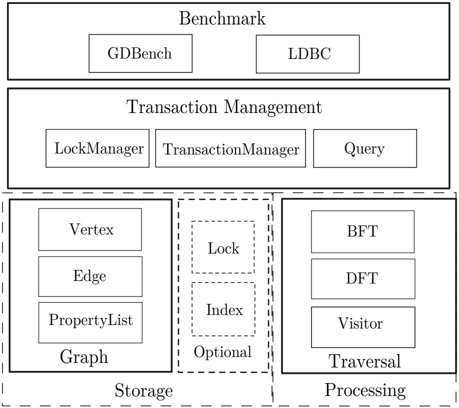
Name	**GDBench**	**LDBC-SNB**
Vertex Types	2	13
Edge Types	2	20
Number of Queries	13	22
Query Example	Get the webpages liked by the friends of a person	Find a person's friends and friends of friends who started working in some company in a given country before a given year
Description	Data generator synthetically generates graphs that model social network activities with different connectivity.	LDBC-SNB models real-life social activities during a period of time.

Example usage of Gromit





Summary



https://git.uwaterloo.ca/caesr-pub/gromit

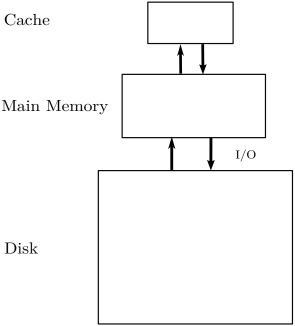
References

- http://cassandra.apache.org/
- https://docs.mongodb.com/manual/
- http://all-free-download.com
- DeCandia et al. 2007. Amazon's Highly Available Key-value Store. *SIGOPS Oper. Syst. Rev.* 41, 6, 205-220.
- Eswaran et al. 1976. The Notions of Consistency and Predicate Locks in a Database System. *Commun, ACM* 19, 11, 624-633.

Thank you

Question?

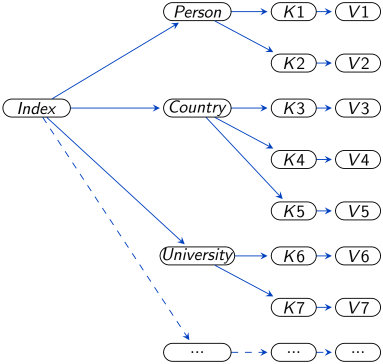
In-Memory or Disk



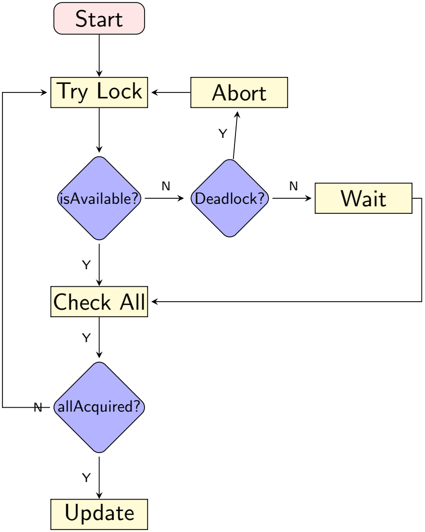
- Processor requests data from cache
- Data in cache are replaced by that from main memory
- Disk stores data and supplies to main memory
- Disk I/O can be bottleneck for memory-intensive workloads

Index

- Indexing retrieves information without traversing
- Indices are grouped by label
- Support for indexing is limited to unique keys in property list, such as ID



Deadlock Prevention Techniques



No-Wait

Never wait for a lock.
Abort right away.

Wait-Die

T_i waits for T_j only if i < j.
Otherwise, abort.

Detection

Construct *Wait-For Graph* and check for cycles. If a cycle exists, abort.
Otherwise, wait.