Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud

Problem:

Existing ***distributed*** graph computation systems perform poorly on **Natural Graphs**.

GraphLab can….

- Expresses asynchronous
- Dynamic, graph-parallel computation
- Ensure data-consistency and high degree of parallelism

- Common properties and Limitations
- Implementing the distributed execution model
- Fault tolerance
- Application Example

2 MLDM Algorithm Properties

- Graph Structured Computation
- Asynchronous Iterative Computation
- Dynamic Computation
- Serializability

2.1 Graph Structured Computation

- Dependency between data
- Product recommendations
- Supported by MapReduce not efficient.

2.1 Graph Structured Computation

For GraphLab……

- Vertex-centric model

(computation is defined as kernels that run on each vertex)

- Sequential shared memory

(each vertex can read and write to data on adjacent vertices and edges)

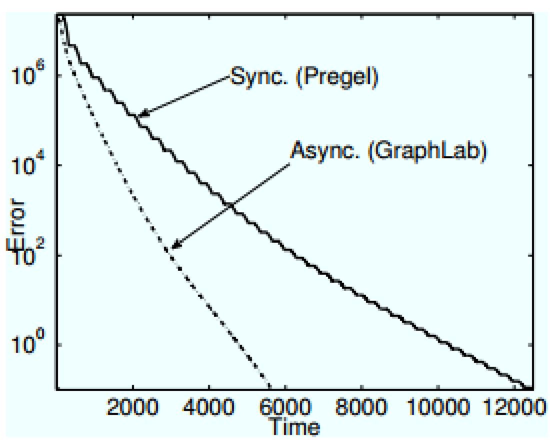
2 MLDM Algorithm Properties

- Graph Structured Computation
- **Asynchronous Iterative Computation**
- Dynamic Computation
- Serializability

2.2 Asynchronous Iterative Computation

- Update using the ***most recent*** parameter values as input
- Providing many MLDM algorithm with benefits

(*Eg*: Linear systems can converge faster)



(a) Async vs Sync PageRank

2.2 Asynchronous Iterative Computation

Disadvantages for synchronous computation:

- Incurs performance penalties

  (slowest machine, load & network imbalance)

- Individual vertex kernels produce more variability

  (Even graph is partitioned uniformly)

2.2 Asynchronous Iterative Computation

For GraphLab……

- Designed to efficiently and naturally express the asynchronous iterative algorithms to advance MLDM

2 MLDM Algorithm Properties

- Graph Structured Computation
- Asynchronous Iterative Computation
- **Dynamic Computation**
- Serializability

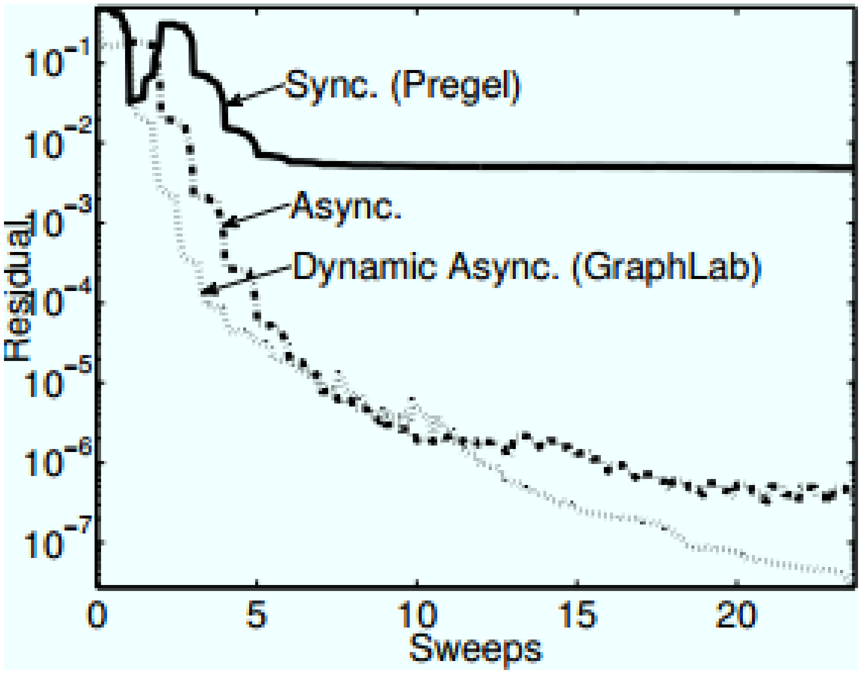
2.3 Dynamic Computation

- In many MLDM alg, iterative computation converges asymmetrically

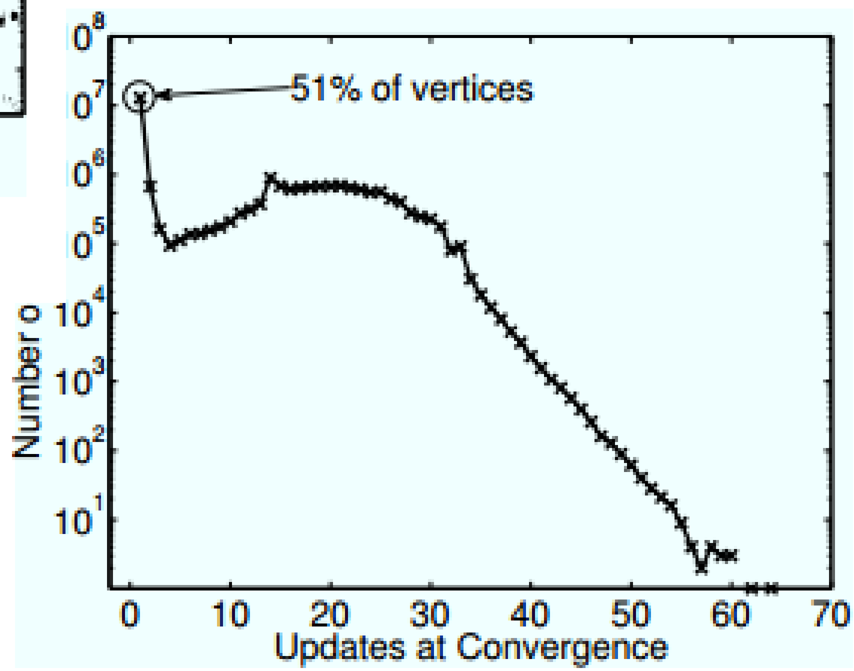
  (*Eg: parameter optimization*)

- Prioritizing computation can further accelerate convergence

  (wasting time if upgrade equally,
  b/c recomputing params converge
  efficiently)



(c) LoopyBP Conv.



(b) Dynamic PageRank

2.3 Dynamic Computation

For GraphLab……

- Only GraphLab permits prioritization as well as the ability to adaptively pull information from adjacent vertices .

2 MLDM Algorithm Properties

- Graph Structured Computation
- Asynchronous Iterative Computation
- Dynamic Computation
- **Serializability**

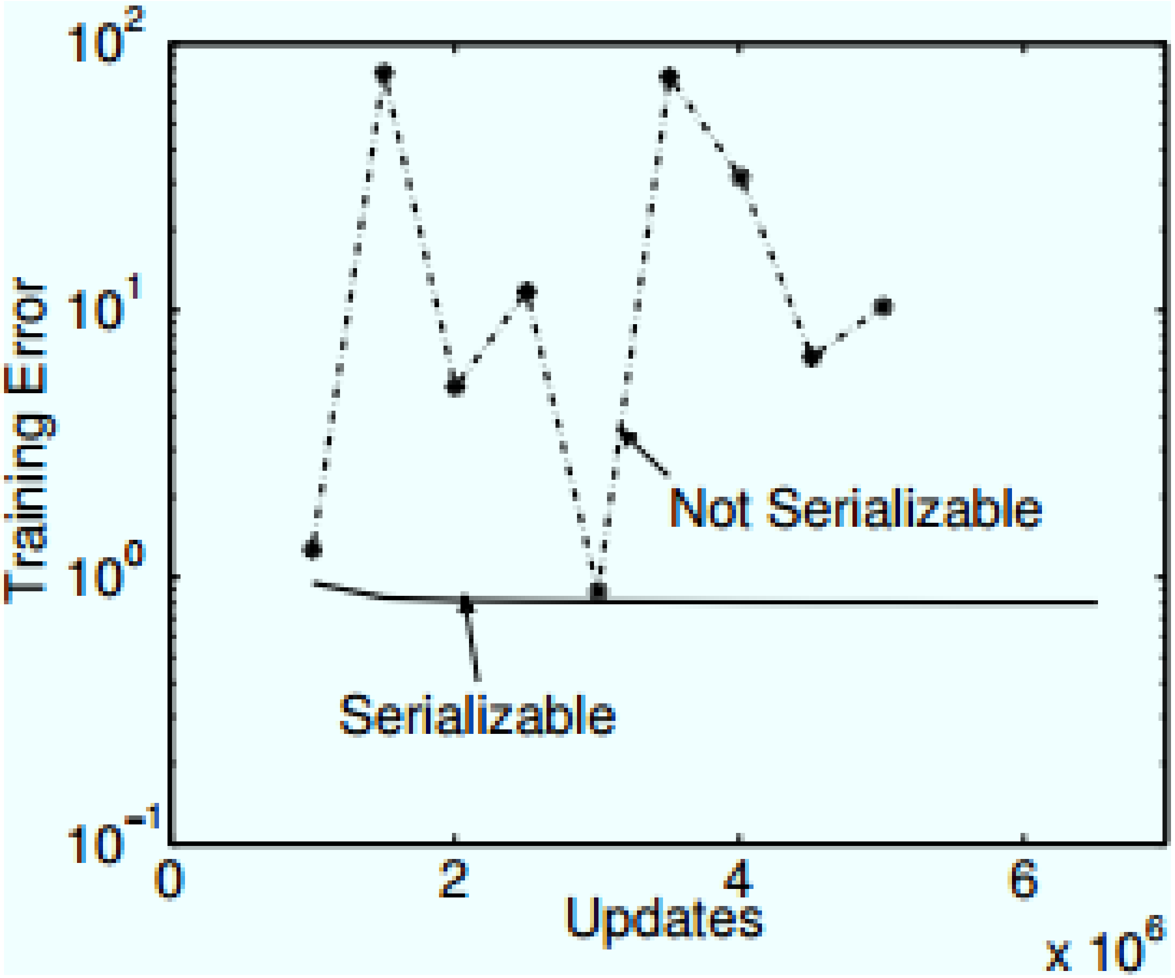
2.4 Serializability

- Eliminates many challenges

  (ensuring all parallel executions have an equivalent sequential execution)

- Converge faster
- Some algorithms require serializability for correctness

  (*Eg:*Dynamic ALS (Sec. 5.1) is unstable when allowed to race)



(d) ALS Consistency

2.4 Serializability

For GraphLab……

- GraphLab supports a broad range of consistency settings, allowing a program to choose the level of consistency needed for correctness

3 Dist. GraphLab Abstraction

- Data Graph
- Update Function
- Sync operation

3 Dist. GraphLab Abstraction

- Data Graph

---- represents program state, and stores both the mutable user-defined data and encodes the sparse computational dependencies

3 Dist. GraphLab Abstraction

- Update function

---- represents the user computation and operate on the data graph

3 Dist. GraphLab Abstraction

- Sync operation

---- represents and maintains global aggregates

Example: PageRank Algorithm

$$R(v) = \frac{a}{n} + (1-a) \sum_{\text{u links to v}} w_{u,v} \times R(u)$$

Rank of webpage v

Weighted sum of neighbors' ranks

- Update ranks in parallel
- Iterate until convergence

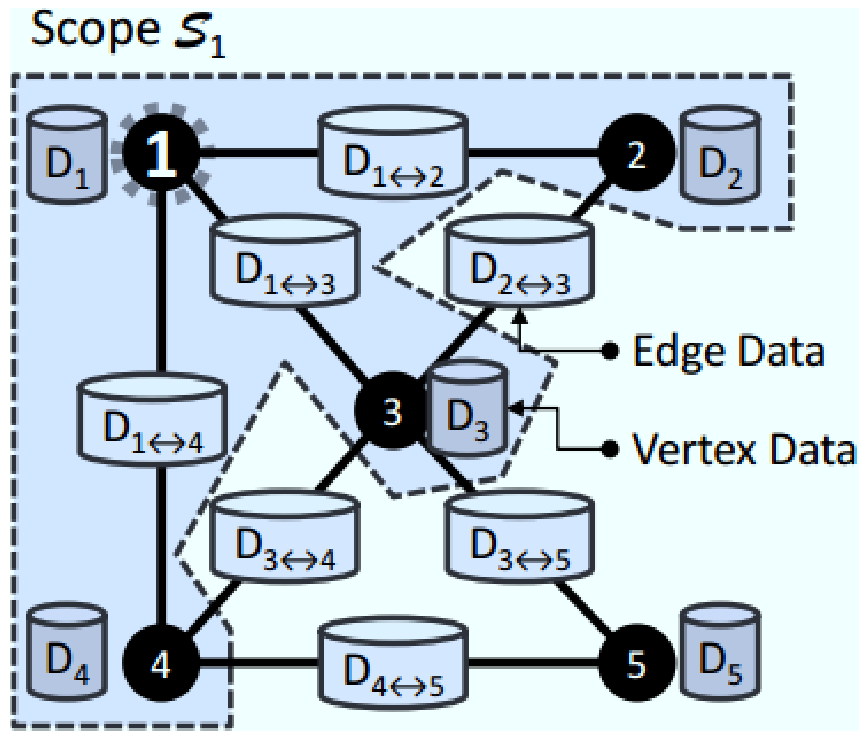
Example: PageRank Algorithm

Part 1 : Data Graph

$D(v)$

$R(v)$

$D_{u \to v}$



(a) Data Graph

Example: PageRank Algorithm

Part 2 : Update Functions

- *A stateless procedure that modifies the data within the scope of a vertex*
- *Schedules the future execution*

$$\text{Update: } f(v, S_v) \rightarrow (S_v, \tau)$$

- *Define update functions with complete freedom*

Example: PageRank Algorithm

Algorithm 1: PageRank update function

Input: Vertex data $\mathbf{R}(v)$ from $\mathcal{S}_v$
Input: Edge data $\{w_{u,v} : u \in \mathbf{N}[v]\}$ from $\mathcal{S}_v$
Input: Neighbor vertex data $\{\mathbf{R}(u) : u \in \mathbf{N}[v]\}$ from $\mathcal{S}_v$
$\mathbf{R}_{\text{old}}(v) \leftarrow \mathbf{R}(v)$ `// Save old PageRank`
$\mathbf{R}(v) \leftarrow \alpha/n$
foreach $u \in \mathbf{N}[v]$ **do** `// Loop over neighbors`
  $\mathbf{R}(v) \leftarrow \mathbf{R}(v) + (1-\alpha) * w_{u,v} * \mathbf{R}(u)$

`// If the PageRank changes sufficiently`
if $|\boldsymbol{R}(v) - \boldsymbol{R}_{old}(v)| > \epsilon$ **then**
  `// Schedule neighbors to be updated`
  return $\{u : u \in \mathbf{N}[v]\}$

Output: Modified scope $\mathcal{S}_v$ with new $\mathbf{R}(v)$

Example: PageRank Algorithm

For GraphLab:

Algorithm 2: GraphLab Execution Model

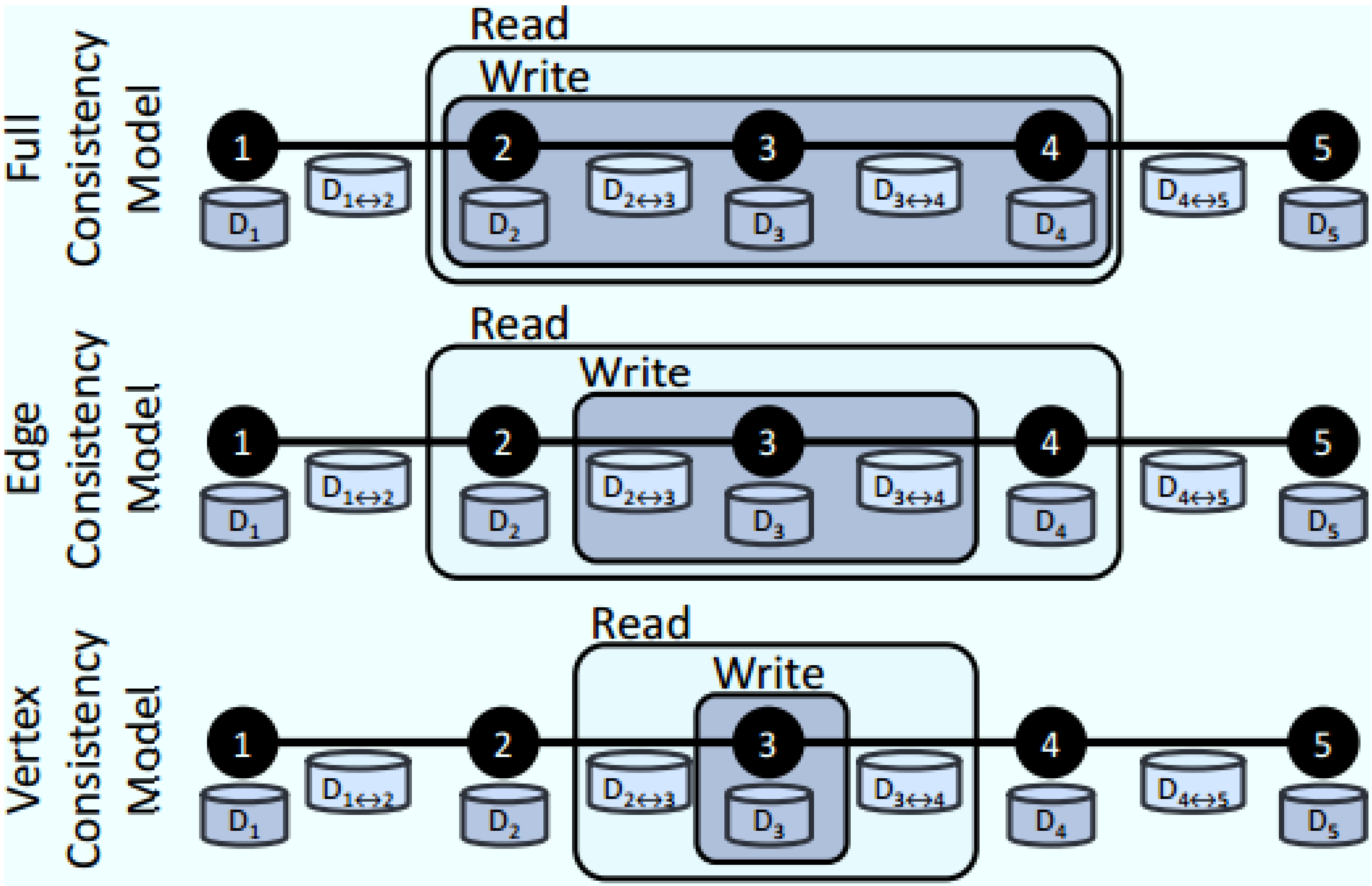
Input: Data Graph $G = (V, E, D)$
Input: Initial vertex set $\mathcal{T} = \{v_1, v_2, ...\}$
while $\mathcal{T}$ *is not Empty* **do**
  $v \leftarrow \texttt{RemoveNext}\,(\mathcal{T})$
  $(\mathcal{T}', \mathcal{S}_v) \leftarrow f(v, \mathcal{S}_v)$
  $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}'$
Output: Modified Data Graph $G = (V, E, D')$

- ***Notice**: Allow runtime to determine the best order to execute vertices*

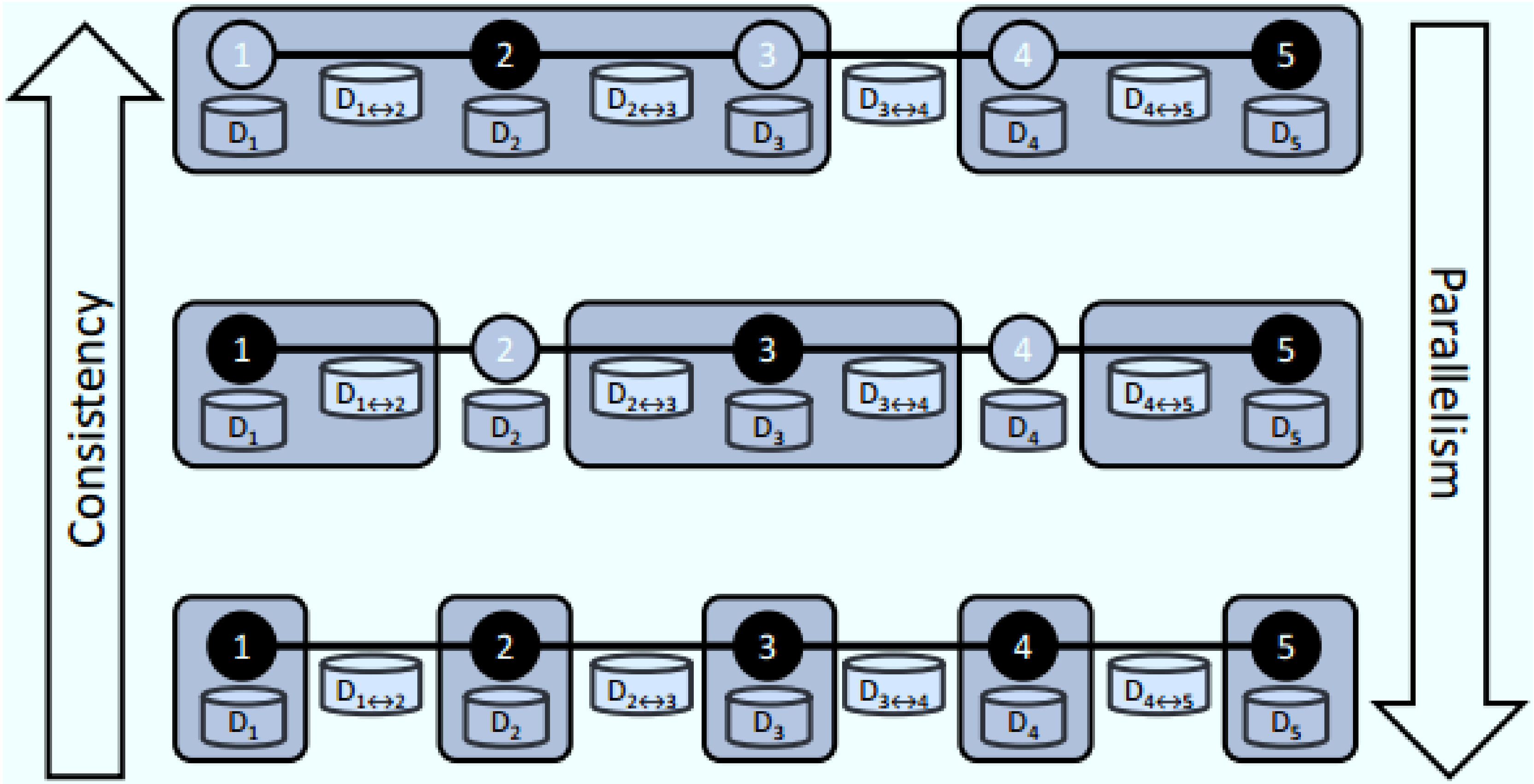
Example: PageRank Algorithm

Ensuring Serializability

- A serializable execution implies that there exists a serial schedule of update functions that when executed by Alg. 2 produces the same values.
- Scope of update functions do not overlap i.e. the ***full consistency model***



(b) Consistency Models



(c) Consistency and Parallelism