Distributed GraphLab

A Framework for Machine Learning and Data Mining in the Cloud

Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, J. Hellerstein

By Maciej Biskupiak for R212

Motivation

Abstractions of parallel computation are necessary.

Current Models such as MapReduce, Dryad or Pregel are too limiting or inefficient for our purposes.

GraphLab Abstraction

GraphLab is:

- Asynchronous: parameter values are not necessarily updated at the same time
- Dynamic: Parameters are not updated equally often
- Serialisable: All parallel executions have an equivalent serial execution (no data races)

It was originally developed for the multicore in memory setting.

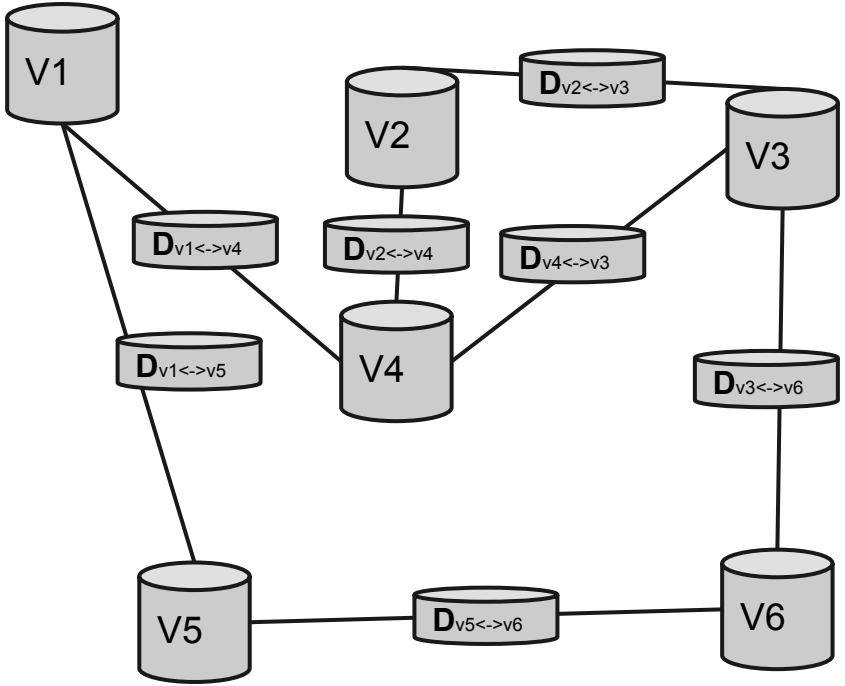
GraphLab Abstraction

GraphLab consists of three main parts:

- The Data Graph
- Update Function
- Sync Function

Data Graph

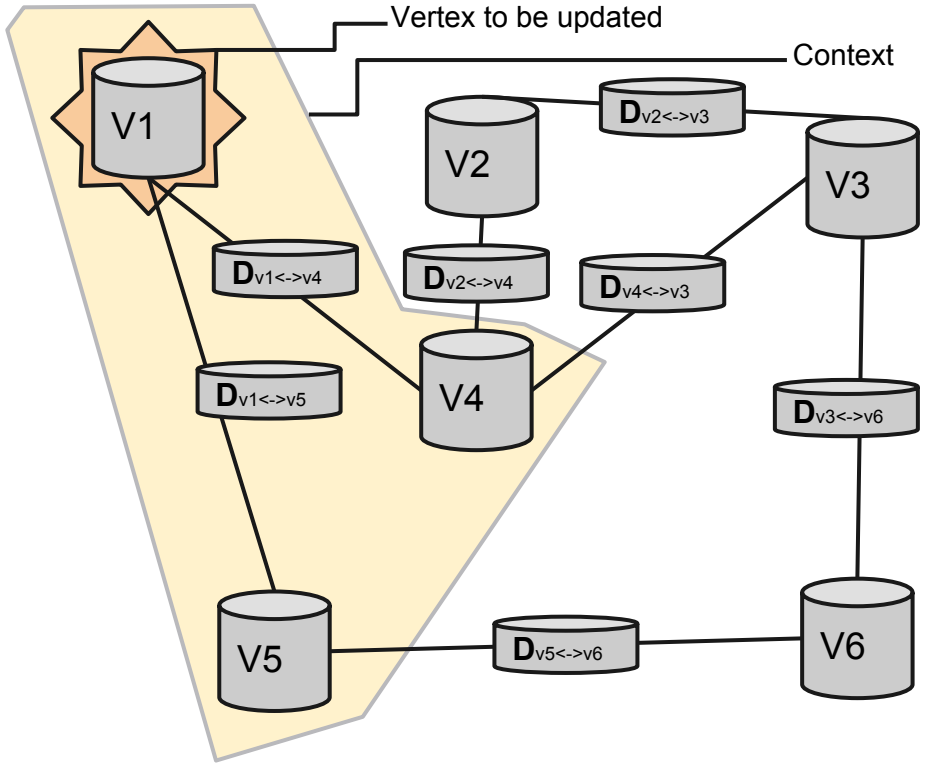
- Computation can be expressed as an arbitrary graph.
- Data is associated either with vertices or edges
- The data itself is mutable, but the structure of the graph is not



Update Function

Takes a vertex *V* and it's surrounding context S_v.

Returns the new values of it's context S_v and a list *T* of vertices that will eventually be updated.



Sync Function

The sync function provides a way to track global state.

Each vertex *v* can publish a global value S_v. The sync function performs an associative sum over all of these values.

Distributed GraphLab

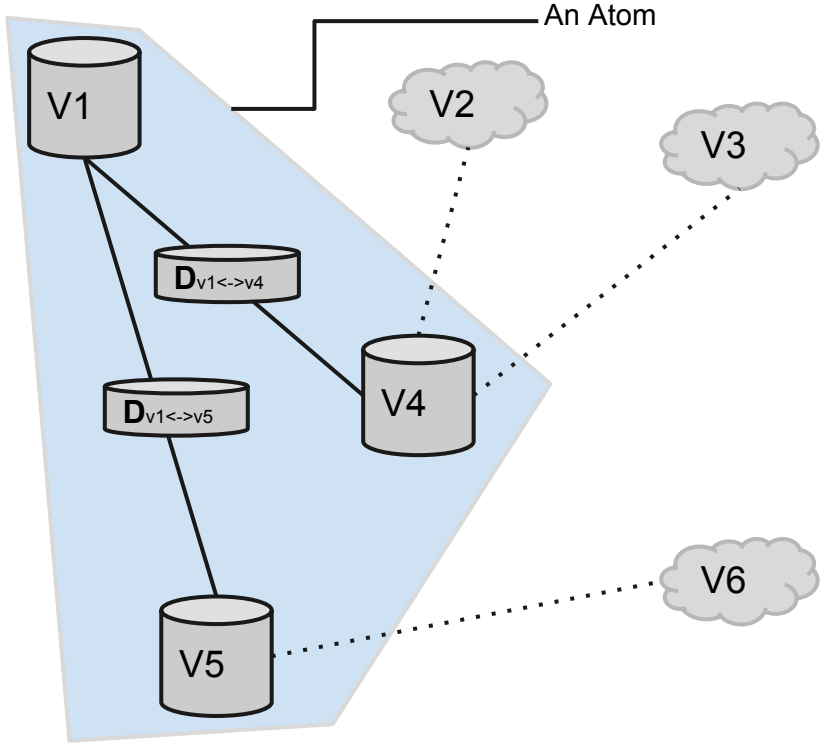
In order to bring GraphLab to the distributed setting, we need a solution for the following:

- Distributing the graph data and balancing the computation
- Maintaining consistency across nodes
- Achieving fault tolerance

Distributing Graph Data

We partition the graph into a set of *K* atoms (where *K* is much greater than the number of servers).

Each atom is stored as a separate file and contains information about ‘ghosts’, the vertices and edges adjacent to the atoms boundary

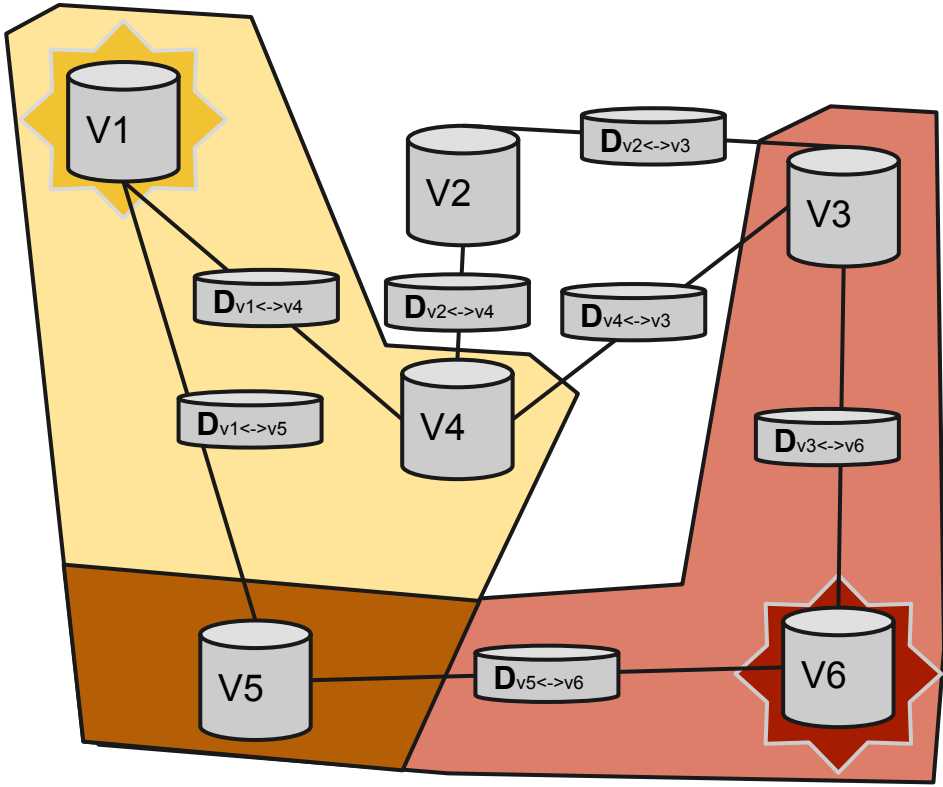


Maintaining Consistency

Data races are possible if the contexts of update functions overlap.

GraphLab provides two means of dealing with this:

- A chromatic engine based on graph coloring
- A distributed read/writer lock system



Levels of consistency

Distributed locking supports various levels of consistency.

- Vertex consistency: Obtains a write lock on the vertex and a read lock on data belonging to adjacent vertices
- Vertex and edge consistency: Obtains a write lock on the vertex and it's adjacent edges and a read lock on it's adjacent vertices
- Total consistency: Obtains a write lock on a vertex and it's adjacent edges and vertices

Gives greater performance, as some problems do not require total consistency (EG. Pagerank)

Fault Tolerance

In the event of a failure the system can recover to a snapshot taken at a previous point.

The snapshot mechanism has to be asynchronous in order to avoid suspending execution.

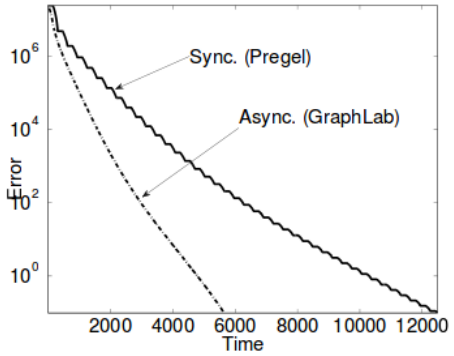
GraphLab implements the Chandy-Lamport algorithm to achieve this

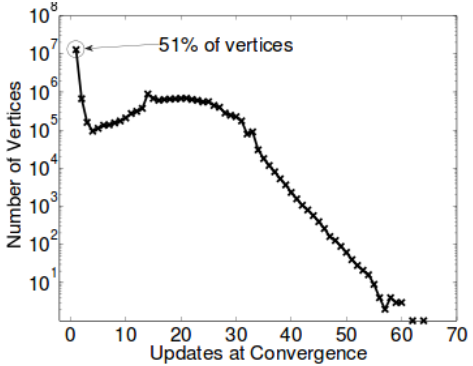
Performance

- Achieves 20-60x improvement over Hadoop
- Competitive with tailored MPI implementations
- Error can converge almost two times faster than in non-dynamic computation

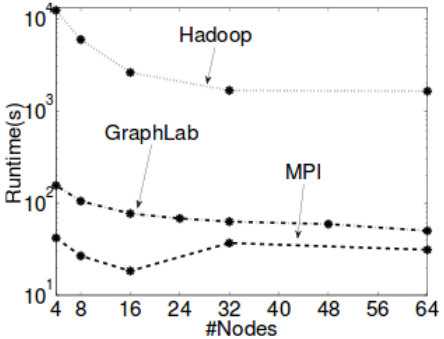
Performance

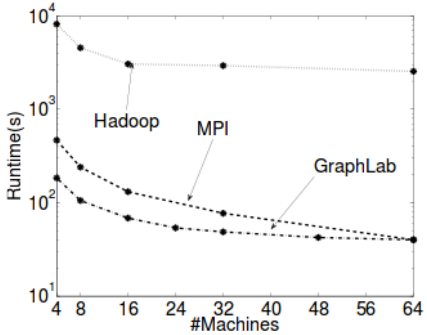
Asynchronous vs synchronous performance of pagerank





Comparison of scalability on Named Entity Recognition (First) and The Netflix Collaborative Filtering (Second)





Conclusion

Powerful abstraction of parallel computation brought to the distributed setting

Provides more flexibility than other models, constrained only by inability to modify the graph structure.