DYNAMOGRAPH: A DISTRIBUTED SYSTEM FOR LARGE-SCALE, TEMPORAL GRAPH PROCESSING, ITS IMPLEMENTATION AND FIRST OBSERVATIONS

Matthias Steinbauer, Gabriele Anderst-Kotsis
Institute of Telecooperation
INTRODUCTION / MOTIVATION

- Graphs serve as models for real world structures in many different disciplines
  - Social sciences > Social networks
  - Biology > Protein-Protein Interactions
  - Cartography > Digital road maps
  - Web > The web graph
- Graph and network models are very well studied in mathematics computer science and related disciplines
- So why is there need for new research in this area?
REAL WORLD GRAPHS OFTEN GROW TO LARGE SCALES.

It is not feasible to process large graphs with traditional tools and algorithms.

Exceed memory size of single computer

require new tactics for visualisation

It is not feasible to process large graphs with traditional tools and algorithms.
THE DIMENSION OF TIME CANNOT BE NEGLECTED

**Static** views on graphs often show **blurred** or **too dense** data.

**Biological** processes are **time** dependent.

**Static reachability** measures in social networks do not hold for dynamic networks.
TEMPORAL GRAPHS

**Graph** $G$ is a pair $(V, E)$ where $V$ denotes the set of vertices and $E$ denotes the set of edges between any $v, e \in V$

A **temporal graph** $T$ can be given as a set of graphs $T = \{G_1, G_2, G_3, \ldots, G_t\}$ where each $G_x = (V_x, E_x)$ $G_x$ is called a **static snapshot** at time $x$ And $G_{tm..tn}$ as a selection of multiple $G_x$ from $T$ is a static snapshot for a **timespan**
Due to popular demand traditional graph databases now support temporal data storage.

Trivial solution is to model time-spans as nodes and connect data nodes only via time-span nodes.

Efficient Querying (Important requirement for a database)

Get all nodes starting with $T_2$

Only distributed / scalable storage but not processing.

---

DISTRIBUTED MATRIX PROCESSING

(Sparse) matrices have long served as model for graphs.

Many popular graph algorithms have efficient implementations for matrices.

Distributed matrix processors (multiplicators) can be used to cope with large scale graphs.

Temporal aspect hard to integrate.

---

DISTRIBUTED GRAPH PROCESSING

- Dominantly implementations on top of Big Data systems
- Data-set is living on a distributed file-system
- Graph processing jobs are implemented as MapReduce jobs
- Famous Google Pregel paper with open source implementations (Giraph, GPS, etc.)
- Temporal aspects not covered

CHRONOS AND IMMORTALGRAPH

- Research line by Microsoft
- Also introduces a concept of time-spans for analysis
- Current iterations designed towards clever **in-memory** layout:
  - **temporal** locality
  - **structural** locality


HORIZONTAL SCALABILITY
TEMPORAL PREGEL IN A SINGLE SLIDE

- A **compute** function \( c \) / **execute** is implemented by the **application developer** which is executed in **iterations / steps**

- It runs in the context of a vertex \( v \), a timespan \( t \), and receives a list of incoming messages \{\( m_0, m_1, ..., m_n \}\)

- \( c \) sees \( v \) as it would have looked in timespan \( t \)

- Since \( c \) relies on **local information** only, it can be executed **in parallel** for all vertices

- Messages can be sent to other vertex threads through a **messaging function** \( m(v, x) \) sending message \( x \) to vertex \( v \)

- Messages **sent** in step \( s \) are **received** in step \( s+1 \)

- The code in \( c \) can **vote to halt** algorithm execution; execution halts if **all vertices** voted to halt; vertices that voted get **disabled**

- Vertices are **enabled** if they **receive messages**
EXTENSIONS OVER THE ORIGINAL PREGEL FRAMEWORK

- **Temporal filtering**: algorithms can be run in the context of a timespan such that only data from this timespan is considered by the framework

- **Execution Triggers**

- **Global memory**: with related management functionality to allow:
  - **Initialisation**: Prior algorithm scheduling the application developer can initialise (Settings, Parameters, etc.)
  - **Access and Update**: Slots see a local copy of the global memory map
  - **Merging**: After each step / iteration conflicts in the global map must be resolved through (user defined) merge strategies
PAGE RANK IN DYNAMOGRAPH

- PageRank is a **natural** fit for Pregel style computation.
- Originally (and perhaps still) used by Google to rate websites for their search index.

Each vertex **starts** out with an **initial PageRank** `INITRANK`.

In a **continuous process** each vertex reads its own PageRank, **divides** it by the **number** of (outgoing) **edges** and sends this number to its **neighbours**.

Vertices **receive** all **messages** from neighbours, and set the **sum** as their **new PageRank**.

---

public void execute(
    List<VertexMsg> messages, VertexContext vertexContext,
    SuperStepContext superstepContext,
    Timeframe timeframe, Vertex vertex) {
    if (this.getStep() >= PageRank.MAX_ITER) {
        voteToHalt(vertexContext); return;
    }
    if (this.getStep() == 0L) {
        setPageRank(vertexContext, PageRank.INITRANK);
        Collection<Edge> outEdges = vertex.getWeightedOutEdgesReading();
        float outRank = (pageRank * PageRank.DAMP) / outEdges.size();
        for (Edge out : outEdges) {
            sendMessage(out.getTarget(), outRank);
        }
    } else {
        if (messages.size() > 0) {
            float changedby = 0.0f; float sumIncoming = 0.0f;
            for (VertexMessage m : messages) {
                sumIncoming += m.getFloatValue();
            }
            changedby = setPageRank(vertexContext, sumIncoming);
            if (changedby >= PageRank.SWING_THRESHOLD) {
                Collection<Edge> outEdges = vertex.getWeightedOutEdgesReading();
                float pageRank = getPageRank(vertexContext);
                float outRank = (pageRank * PageRank.DAMP) / outEdges.size();
                for (Edge out : outEdges) {
                    sendMessage(out.getTarget(), outRank);
                }
            }
        }
        voteToHalt(vertexContext);
    }
}
LABEL PROPAGATION COMMUNITY DETECTION IN DYNAMOGRAPH

Observe neighbouring community labels
Apply the label most often found or
if none such can be found the smallest one

LABEL PROPAGATION COMMUNITY DETECTION IN DYNAMOGRAPH

Observe neighbouring community labels
Apply the label most often found or
if none such can be found the smallest one
LABEL PROPAGATION COMMUNITY DETECTION IN DYNAMOGRAPH

Observe neighbouring community labels
Apply the label most often found or
if none such can be found the smallest one
LABEL PROPAGATION COMMUNITY DETECTION IN DYNAMOGRAPH

after some more iterations

Observe neighbouring community labels
Apply the label most often found or
  if none such can be found the smallest one
public void execute(
    List<VertexMsg> messages, VertexContext vertexContext,
    SuperStepContext superstepContext,
    Timeframe timeframe, Vertex vertex) {
    if (this.getStep() == 0L) {
        this.setVertexLabel(vertexContext, vertex.getId());
        propagateVertexLabel(vertex, vertexContext);
    } else if (this.getStep() >= 8) { // 8 iterations
        voteToHalt(vertexContext);
    } else {
        Map<Long, Integer> count = new HashMap<Long, Integer>();
        for (VertexMessage m : messages) {
            Long clusterId = (Long) m.getBody();
            if (!count.containsKey(clusterId)) {
                count.put(clusterId, 1);
            } else {
                Integer currentCount = count.get(clusterId);
                currentCount++;
                count.put(clusterId, currentCount);
            }
        }
        long clusterId = findMaxClusterId(count);
        setVertexLabel(vertexContext, maxClusterId);
        propagateVertexLabel(node, vertexContext);
    }
}
}
967 members
IRC channel

full time-span
967 members IRC channel

selected time-span

1\textsuperscript{st} June 2008
15\textsuperscript{th} June 2008
REAL WORLD SOCIAL NETWORKS

- Data is available as online resource
- Users can filter and load data into DynamoGraph
- Algorithms for automatic layout of the visualisation are available (ForceAtlas2)

In visualisation it is already clear that reachability measures computed on the full time-span will often not hold on shorter time-spans

- How is information dissemination influenced by that fact?
- Can we perform cluster analysis on users and identify topics of interest?
FUTURE WORK

- More temporal graph algorithms are to be implemented (reachability, clustering, ...) to provide more interesting metrics to our users
- Proper Evaluation targeted Dataset: Click from Indiana University
- Private Cloud test with 24 CPUs ~288 GB of memory
- MACH super computing infrastructure single system image machine (2048x4 cores, 16 TB memory)
CONCLUSIONS AND FUTURE WORK

- **DynamoGraph** as a platform for large-scale temporal graph processing has matured enough to be evaluated in scientific scenarios.

- Two example algorithms were shown that demonstrate how potential application developers can create processing pipelines in this paradigm.

- A web-based graph work-bench as a first prototype was briefly shown to demonstrate the necessity of temporal analysis.
Matthias Steinbauer
matthias.steinbauer@jku.at

slides available at
http://steinbauer.org/