Towards Effective Partition Management for Large Graphs

Shengqi Yang, Xifeng Yan, Bo Zong and Arijit Khan (UC Santa Barbara)
Presenter: Xiao Meng
Motivation
- How to manage large graphs?

- Increasing demand for large graph management on commodity servers
  ✓ Facebook: 890 million daily active users on average for December 2014

- Achieving fast query response time and high throughput
  ✓ Partitioning/distributing and parallel processing of graph data
  ✓ However... It’s always easier said than done.
Outline

- Background
- Overview of Sedge
- Techniques of Sedge
  - Complementary partitioning
  - On-demand partitioning
  - Two-level partition management
- A Look Back & Around
- Experimental Evaluations
- Conclusions & Takeaways
- Q & A
Background
- Solutions available

- Memory-based solution
  - Single-machine: Neo4j, HyperGraphDB
  - Distributed: Trinity [1]

- General distributed solution
  - MapReduce-style ill-suited for graph processing

- More specialized solution
  - Graph partitioning and distribution
  - Pregel [2], SPAR [3]
Background
- Graph query workload types

- Queries with random access or complete traversal of an entire graph
- Queries with access bounded by partition boundaries
- Queries with access crossing the partition boundaries

Figure taken from "Towards Effective Partition Management for Large Graphs", SIGMOD 2012
Overview of Sedge
- Self Evolving Distributed Graph Management Environment

- Built upon Pregel, but eliminating constraints and solving problems facing it
  - Workload balancing, overhead reduction, duplicate vertices...
- Leveraging partitioning techniques to achieve that
  - 2-level partition architecture supports complementary and on-demand partitioning

Figure taken from “Towards Effective Partition Management for Large Graphs”, SIGMOD 2012
Techniques of Sedge
- Complementary partitioning

- Idea: repartition the graph with region constraint
- Basically, we want to find a new partition set of the same graph so that the originally cross-partition edges become internal ones

Figure taken from “Towards Effective Partition Management for Large Graphs”, SIGMOD 2012
Techniques of Sedge - Complementary partitioning

- How it’s done theoretically?
  ✓ Formulation to a nonconvex quadratically constrained quadratic integer program (QCQIP) to reuse the existing balanced partitioning algorithms

- How it’s done practically?
  ✓ Solution1: Increase the weight of cut edges by \( \lambda \) then rerun
  ✓ Solution2: Delete all cut edges first then rerun

- How it works then?
  ✓ There could be several partitions capable of handling a query to a vertex \( u \)
  ✓ Queries should be routed to a safe partition: \( u \) far away from partition boundaries
Techniques of Sedge - On-demand partitioning

- Hotspot is a real bummer and it comes in two shapes
  - Internal hotspots located in one partition
  - Cross-partition hotspots on the boundaries of multiple partitions
Techniques of Sedge
- On-demand partitioning

- Hotspot is a real bummer and it comes in two shapes
  - Internal hotspots located in one partition
  - Cross-partition hotspots on the boundaries of multiple partitions

- To deal with internal hotspots: Partition Replication
- To deal with cross-partition hotspots: Dynamic Partitioning
Techniques of Sedge
- On-demand partitioning

- Partition workload: internal, external (cross-partition)
- Partition Replication starts when internal workload is intensive
  - Replicate partition P to P'
  - Send P' to idle machine with free memory space
  - Else replace a slack partition with P'
Techniques of Sedge - On-demand partitioning

- For cross-partition hotspots: Dynamic Partitioning
  - Better to generate new partitions that only cover these areas
  - New partitions only share heavy workload while reduce communication

- Step 1: hotspot analysis
  - Calculate ratio $r = \frac{|W_{ext}(P)|}{|W_{int}(P)| + |W_{ext}(P)|}$
  - $p = \frac{|E_{ext}(P)|}{|E_{int}(P)| + |E_{ext}(P)|}$
  - Hypothesis testing: if $r$ is much higher than $p$, then assume there are cross-partition hotspots in $P$
Techniques of Sedge - On-demand partitioning

- Step 2: Track cross-partition queries
  - Mark the search path with color-blocks
  - Profile a query to an envelope
  - Collect the envelopes to form one new partition

- Color-blocks: coarse-granularity units to trace path of cross-partition queries

- Envelope: a sequence of blocks that covers a cross-partition query

- Envelope Collection: put the maximized # of envelopes into a new partition wrt. space constraint

Figure taken from "Towards Effective Partition Management for Large Graphs", SIGMOD 2012
Techniques of Sedge
- On-demand partitioning

- Envelope collection objective
  - Put the maximized # of envelopes into a new partition wrt. size constraint
  - A classic NP-complete problem: Set-Union Knapsack Problem

- A greedy algorithm to save the day
  - Intuition: combining similar envelopes consumes less space than combining non-similar ones
  - Metric: Jaccard coefficient \( Sim(L_i, L_j) = \frac{|L_i \cap L_j|}{|L_i \cup L_j|} \)
  - Solution: Locality-sensitive Hashing
Techniques of Sedge
- On-demand partitioning

- Envelope collection objective
  ✓ Put the maximized # of envelopes into a new partition wrt. size constraint
  ✓ A classic NP-complete problem: Set-Union Knapsack Problem

✓ A greedy algorithm to save the day
✓ Intuition: combining similar envelopes consumes less space than combining non-similar ones
✓ Metric: Jaccard coefficient \( \text{Sim}(L_i, L_j) = \frac{|L_i \cap L_j|}{|L_i \cup L_j|} \)
✓ Solution: Locality-sensitive Hashing – Min-Hash
Techniques of Sedge - On-demand partitioning

- **Step 2: Track cross-partition queries**
  - Mark the search path with color-blocks
  - Profile a query to an envelope
  - Collect the envelopes to form one new partition

- **Step 3: Partition Generation**
  - Assign each cluster a benefit score \( \rho = \frac{|W(C)|}{|C|} \)
  - Iteratively add the cluster with the highest \( \rho \) to an initially empty partition (as long as the total size \( \leq \) the default partition size \( M \))
Techniques of Sedge
- On-demand partitioning

- Step 2: Track cross-partition queries
  - Mark the search path with color-blocks
  - Profile a query to an envelope
  - Collect the envelopes to form one new partition

- Step 3: Partition Generation
  - Assign each cluster a benefit score $\rho = \frac{|W(C)|}{|C|}$
  - Iteratively add the cluster with the highest $\rho$ to an initially empty partition (as long as the total size \( \leq \) the default partition size $M$)

- Discussion: too good to be true?
Techniques of Sedge - Two-level partition management

- Two-level partition architecture
  - Primary partitions: A, B, C and D inter-connected in two-way
  - Secondary partitions: B’ and E connected with primary ones in one-way

Figure taken from “Towards Effective Partition Management for Large Graphs”, SIGMOD 2012
A Look Back & Around
- Other modules of Sedge

- meta-data manager

✓ Meta-data maintained by master and Pregel instances (PI)

✓ In master: info about each PI and a table mapping vertices to PI

✓ (Instance Workload Table, Vertex-Instance Fitness List)

✓ In PIs: an index mapping vertices to partitions in each PI

✓ (Partition Workload Table, Vertex-Primary Partition Table, Partition-Replicates Table, Vertex-Dynamic Partitions Table)

Figure taken from "Towards Effective Partition Management for Large Graphs", SIGMOD 2012
A Look Back & Around
- Other modules of Sedge

- Performance Optimizer
- Continuously collects run-time information from all the PIs and characterizes the execution of the query workload
- The master updates IWT while PIs maintain the PWTs

Figure taken from "Towards Effective Partition Management for Large Graphs", SIGMOD 2012
A Look Back & Around
- Other related works

- Large-scale graph partitioning tools
  ✓ METIS, Chaco, SCOTCH

- Graph platforms
  ✓ SHS, PEGASUS, COSI, SPAR

- Distributed query processing
  ✓ Semi-structured, relational, RDF data
Experimental Evaluations - With RDF Benchmark

Hardware setting
- 31 computing nodes
- One serves as the master and the rest workers

$SP^2$Bench
- Choose the DBLP library as its simulation basis
- 100M edges with 5 Queries: Q2, Q4, Q6, Q7, Q8
Experimental Evaluations
- With RDF Benchmark

- **Experiment setting**
  - Partition configuration: CP1 to CP5
  - Workload: 10,000 random queries with random starts

- **Results**
  - Significant cross-partition query reduction
  - Cross-partition query vanishes for Q2, Q4 and Q6

Figure taken from “Towards Effective Partition Management for Large Graphs”, SIGMOD 2012
Experimental Evaluations - With RDF Benchmark

- **Experiment setting**
  - Partition Configuration: CP1*5, CP5 and CP4+DP
  - Evolving query workload: evolving 10,000 queries with 10 timestamps

- **Results**
  - Blue vs. green: effect of complementary partitioning
  - Green vs. red: effect of on-demand partitioning

Figure taken from "Towards Effective Partition Management for Large Graphs", SIGMOD 2012
Experimental Evaluations - With Real Graph Datasets

Datasets

<table>
<thead>
<tr>
<th>Graph</th>
<th>Size (GB)</th>
<th>Partition (s)</th>
<th>VFL (MB)</th>
<th>VPT (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>14.8</td>
<td>120</td>
<td>81.5</td>
<td>35.3</td>
</tr>
<tr>
<td>Twitter</td>
<td>24</td>
<td>180</td>
<td>109.0</td>
<td>45.4</td>
</tr>
<tr>
<td>Bio</td>
<td>13</td>
<td>40</td>
<td>135.9</td>
<td>55.3</td>
</tr>
<tr>
<td>Syn.</td>
<td>17</td>
<td>800</td>
<td>543.7</td>
<td>205</td>
</tr>
</tbody>
</table>

Query workload

- neighbor search
- random walk
- random walk with restart
Experimental Evaluations
- With Real Graph Datasets

Complementary Partitioning

Partition replication: throughput

Dynamic Partitioning: runtime cost

Dynamic partitioning: response time

Cross-partition queries vs. Improvement ratio in avg. response time

Figure taken from “Towards Effective Partition Management for Large Graphs”, SIGMOD 2012
Conclusions & Takeaways

- Partitioning techniques with two level partition management
  - Complementary partitioning
  - On-demand partitioning
- Greedy algorithm for dynamic partitioning

- Takeaways:
  - One partition scheme cannot fit all
  - Always a tradeoff between data locality and load balancing
  - Future work can be done regarding efficient distributed RDF data storage management, distributed query processing over RDF, etc.
1. In this work, a major assumption is that the network bandwidth is consistent for each pair of nodes. But in reality, it’s often not the case. How to efficiently manage partitions in a distributed setting with network bandwidth unevenness?

2. Metadata are becoming big data as well. In this design, the VPT is a few GB for each node. In estimation, metadata is 0.1% - 1% of the data space [4]. How to efficiently manage these tables? More generally, how to efficiently manage graph metadata?

3. How to compare or extend Sedge to other settings and partition metrics:
   - Setting: multi-processors?
   - Data model: hyper-graph?
   - Metrics: Query makespan or boundary cut?
References


Backup
- Duplicate sensitive graph query

- Use UNION instead of SUM.