GraphTrek: Asynchronous Graph Traversal for Property Graph-Based Metadata Management

Dong Dai, Philip Carns, Robert B. Ross, John Jenkins, Kyle Blauer, and Yong Chen
Metadata Management Challenges in HPC

- Many management functionalities are needed in HPC
  - Results Validation; Data Audit; Provenance Recording; Flexible Data Organization

- POSIX metadata is not enough, we need to:
  - Know how users start an execution, including large workflows, parallel jobs, or local processes.
  - Know how executions access files, both on reading and writing.

- We described HPC rich metadata as:
  - detailed information about more entities and their relationships
  - arbitrary user-defined entities, relationships, and attributes.
An Example: Flexible Data Organization

• Usually, we organize data files, configuration files, and results from the same project together.

• Then
  • What about to check all the inputs or outputs of a single simulation run?
An Example: Flexible Data Organization

• Usually, we organize data files, configuration files, and results from the same project together.

• Then

  • What about to check all the inputs or outputs of a single simulation run?
An Example: Flexible Data Organization

- Usually, we organize data files, configuration files, and results from the same project together.
- Then
  - What about to check all the inputs or outputs of a single simulation run?

Tree-based model from POSIX is limited for this job
Graph-based Metadata Model

- We proposed graph-based model for HPC metadata
Graph-based Metadata Model

• We proposed graph-based model for HPC metadata
Graph-based Metadata Model

- We proposed graph-based model for HPC metadata
Travel on Graph

• With the graph data model,
  • to locate a data file needs to travel the graph
  • to audit users indicates traversal from users to all its execution history
  • to validate results needs to trace back from outputs to their initial inputs

• Graph traversal is a key to effectively answer metadata queries
BFS Graph Traversal

- “Level-synchronous” breadth-first search (BFS)
- Commonly used distributed graph traversal solution
- Used in many graph databases and graph processing frameworks

http://giraph.apache.org/
BFS Graph Traversal

• It faces **straggler problem** as it has global synchronization between steps.

• Metadata graphs simply make this **straggler problem** worse
  
  • As a metadata service, it has to serve concurrent traversal requests

  • Metadata graphs are following power-law distribution. This leads to uneven loads indicating more stragglers

  • Metadata graph traversal can be really deep. It can easily exceed graph diameter
Metadata Graph Traversal Patterns

- Operations on metadata graph have their own patterns
  - The traversal starts from a set of vertices and travels in steps.
  - In each step, it filters vertices and vertices according to the attributes.
  - The traversal may revisit the same vertex again in different steps on different attributes.
  - The traversal may return the intermediate vertices instead of the destiny.
GraphTrek Traversal Language

• Lots of traversal languages already
  • SPARQL, GraphGrep, Cypher, Gremlin, Quasar, SQL
• Based on Gremlin, we define an iterative query-building language.
• The core functions:
  • Vertex/Edge Selector: v(), e()
  • Property filter: va() and ea()
  • Return indicator: rtn()
GraphTrek Traversal Language Examples

• Data Auditing
  Find all text files read by “userA” within a timeframe

    1  GTravel.v(userA).e('run')
       .ea('start_ts', RANGE, [t_s, t_e])
    3  .e('read')
       .va('type', EQ, 'text'). rtn()

• Provenance Support
  Find the executions with the model A and inputs have annotation as B

    2  GTravel.v().va('type', EQ, 'Execution').rtn()
    3  .va('model', EQ, 'A')
       .e('read')
    4  .va('annotation', EQ, 'B')
GraphTrek: Implementation
GraphTrek Traversal Submission

- GraphTrek gathers multiple steps into a single batch to submit

(a) Client-Side Traversal

(b) Server-Side Traversal
GraphTrek Asynchronous Execution

• The whole traversal is scheduled on server-side
GraphTrek Traversal Status and Progress Trace

- Asynchronous execution means almost impossible to maintain global status.

- GraphTrek introduces a status-tracing mechanism to identify failures and track progress.
  - We log the creation and termination events of executions in coordinator server
  - If any execution was logged as creation, but did not terminate, we will consider it failed.

- The count of terminated executions over the active executions in each step simply tells us the progress.
GraphTrek Traversal Return

- Traversal stops when it reaches the last step of GTravel instance.
- It can return not only the terminated vertices, but also intermediate vertices in the travel path.
GraphTrek Traversal Optimizations

- Drawback of asynchronous traversal is the redundant vertex visit.
- Traversal-affiliate cache.
  - Preallocate buffer
  - Cache current executions with ID as: \{travel-id, current-step, vertex-id\}
  - Reuse cached results
  - Substitute results with older version
GraphTrek Traversal Optimizations

- Execution Merging and Scheduling
  - Always schedule executions with smaller version to help slower one to catch up
  - Merge request on the same vertex from different versions to save the disk access
Implementation Discussion

• We implement GraphTrek as a traversal engine working with the underlying storage systems.

  • http://discl.cs.ttu.edu/gitlab/dongdai/graphfs

• Graph partitioning makes huge difference on the traversal performance
  • We only consider the most commonly used edge-cut
  • Vertex-cut and more complex partitioning algorithms will be our future work

• To be fair, we compare with BFS-based traversal strategy sharing the same implementation basis of GraphTrek.
GraphTrek: Evaluations
Evaluations

• Software Configurations
  • GraphFS as the underlying storage system
  • DB files are stored on top of GPFS
    • less than 10% performance downgrade
  • RMAT graph with parameters: \( a = 0.45, b = 0.15, c = 0.15, \) and \( d = 0.25 \)

• Hardware Configurations
  • All evaluations are conducted on the Fusion Cluster
  • Each node has a dual-socket, quad-core 2.53 GHz Intel Xeon CPU with 36 GB memory and 250 GB local hard disks.
  • All nodes are connected by high-speed network interconnection (InfiniBand QDR 4 GB/s per link per direction)
  • The global file system contain a 90 TB GPFS
Sensitivity to Optimizations

- Evaluate affect of optimizations for asynchronous traversal executions.
- Three scenarios are compared:
  - Sync-GT: synchronous graph traversal
  - Async-GT: asynchronous graph traversal without optimizations
  - GraphTrek

<table>
<thead>
<tr>
<th>No. Servers</th>
<th>Sync-GT</th>
<th>Async-GT</th>
<th>GraphTrek</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>47.8 s</td>
<td>63.7 s</td>
<td>45.2 s</td>
</tr>
<tr>
<td>4</td>
<td>28.5 s</td>
<td>33.1 s</td>
<td>22.5 s</td>
</tr>
<tr>
<td>8</td>
<td>17.1 s</td>
<td>20.6 s</td>
<td>13.4 s</td>
</tr>
<tr>
<td>16</td>
<td>10.3 s</td>
<td>12.1 s</td>
<td>8.3 s</td>
</tr>
<tr>
<td>32</td>
<td>7.2 s</td>
<td>7.4 s</td>
<td>5.6 s</td>
</tr>
</tbody>
</table>
Sensitivity to Optimizations

- **Detailed Statistics**
  - *redundant visits*: the number of repeated vertex requests detected by the traversal-affiliate caching
  - *combined visits*: the number of repeated vertex requests detected by the traversal-affiliate caching;
  - *real I/O visits*: real vertex accesses to backend storage systems
Synthetic Workloads Evaluation

Use synthetic RMAT-1 to create graphs to test the performance of traversals with different numbers of steps.

2-step

<table>
<thead>
<tr>
<th>Number of Servers</th>
<th>Time Cost (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Sync–GT, GraphTrek</td>
</tr>
<tr>
<td>4</td>
<td>Sync–GT, GraphTrek</td>
</tr>
<tr>
<td>8</td>
<td>Sync–GT, GraphTrek</td>
</tr>
<tr>
<td>16</td>
<td>Sync–GT, GraphTrek</td>
</tr>
<tr>
<td>32</td>
<td>Sync–GT, GraphTrek</td>
</tr>
</tbody>
</table>

4-step

<table>
<thead>
<tr>
<th>Number of Servers</th>
<th>Time Cost (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Sync–GT, GraphTrek</td>
</tr>
<tr>
<td>4</td>
<td>Sync–GT, GraphTrek</td>
</tr>
<tr>
<td>8</td>
<td>Sync–GT, GraphTrek</td>
</tr>
<tr>
<td>16</td>
<td>Sync–GT, GraphTrek</td>
</tr>
<tr>
<td>32</td>
<td>Sync–GT, GraphTrek</td>
</tr>
</tbody>
</table>

8-step

<table>
<thead>
<tr>
<th>Number of Servers</th>
<th>Time Cost (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Sync–GT, GraphTrek</td>
</tr>
<tr>
<td>4</td>
<td>Sync–GT, GraphTrek</td>
</tr>
<tr>
<td>8</td>
<td>Sync–GT, GraphTrek</td>
</tr>
<tr>
<td>16</td>
<td>Sync–GT, GraphTrek</td>
</tr>
<tr>
<td>32</td>
<td>Sync–GT, GraphTrek</td>
</tr>
</tbody>
</table>
Synthetic Workloads with External Interference

- Evaluation of the benefits from asynchronous graph traversal.
  - Insert fixed (50ms) delay into certain number of individual vertex access
  - Different delays are created at different time of traversal
HPC Metadata Management Workloads

- A real world use case
  - Build a metadata graph based on Darshah trace from the Intrepid supercomputers
  - Assume we want to analyze the influence of a suspicious user on the system
  - The GraphTrek script is like this

```python
GTravel.v(suspectUser).e('run')
2   .ea('ts', RANGE, [ts, te])  // select jobs
   .e('hasExecutions')  // select executions
4   .e('write')  // select outputs
   .e('readBy')  // select executions
6   .e('write').rtn();  // outputs of executions
```
HPC Metadata Management Workloads

• A real world use case
  • Build a metadata graph based on Darshan trace from the Intrepid supercomputers
  • Assume we want to analyze the influence of a suspicious user on the system
  • The GraphTrek script is like this

<table>
<thead>
<tr>
<th>No. Servers</th>
<th>Sync-GT</th>
<th>Async-GT</th>
<th>GraphTrek</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>3575 ms</td>
<td>4159 ms</td>
<td>2839 ms</td>
</tr>
</tbody>
</table>
Thanks & Questions
ACKNOWLEDGMENT

Many thanks to the reviewers for their great comments.

This material is based upon work supported by the U.S. Department of Defense; by the U.S. Department of Energy, Office of Science, under Contract No. DE-AC02-06CH11357; and by the National Science Foundation under grant CCF-1409946 and CNS-1338078.