TTC'18: Hawk solution

Answering queries with the Neo4j graph database

What is Hawk?

- Hawk is a heterogeneous model indexing framework:
  - Designed to run queries over many model files
  - In this case we only have one :-(
- Mirrors and links all the models into a graph database
  - We currently support Neo4j, OrientDB, Greycat
  - Always disk-based for now (in-memory DBs later?)
- Provides a DB-agnostic query language
  - Epsilon Object Language
- Can quickly find model elements by:
  - Attribute value (indexed attributes)
  - Expression value (derived attributes/edges)

Solutions implemented

- Naive update + query
- Optimised update + naive query
- Optimised update + optimised query

Solutions implemented: naive solution

- Initialize:
  - Set up Neo4j
  - Register metamodels into Neo4j
  - Register derived attributes
- Load: mirror initial.xmi into Neo4j
- Initial view: run query in EOL
- Update:
  - Load changeX.xmi + initial.xmi
  - Run EOL script to update and save initial.xmi
  - Run incremental reindex of initial.xmi
  - Re-run query in EOL

EMF trickery so we load initial.xmi in reasonable time for sizes > 64

```
10 public class IntrinsicIDEMFModelResourceFactory extends EMFModelResourceFactory {
11
12     @Override
13     protected Factory createResourceFactory() {
14         return new IntrinsicIDXMIResourceFactoryImpl();
15     }
16
17     @Override
18     protected Map<?, ?> createEMFLoadOptions() {
19         return Collections.singletonMap(XMIResource.OPTION_DEFER_IDREF_RESOLUTION, true);
20     }
21
22 }
23
```

```
10 class IntrinsicIDXMIResourceFactoryImpl extends XMIResourceFactoryImpl {
11     @Override
12     public Resource createResource(URI uri) {
13         final XMIResourceImpl r = new XMIResourceImpl(uri);
14         r.setIntrinsicIDToEObjectMap(new HashMap<>());
15         return r;
16     }
17 }
```

Derived attributes: extending types with precomputed expressions

- We can pre-compute the scores for each element
- Scores will be updated incrementally when the nodes they depended on change
- Here we extend Post for Q1 scoring

```
1 // operation Submission score() : Integer {
2   return self.closure(p|p.comments)
3     .flatten
4     .collect(p | 10 + p.likedBy.size)
5     .sum();
6
```

Derived attributes: use within queries

- We can then use it as a regular attribute
- Had to implement a specific Comparator to sort results by score + resolve ties by timestamp
- EOL does not support lambdas

```
1 var scored = Post.all.collect(p|
2   Sequence {p.id, p.score, p.timestamp}
3 ).asSequence;
4 Native('java.util.Collections').sort(
5   scored,
6   new Native('org.hawk.ttc2018.queries.ResultComparator'));
7 return scored.subList(0, scored.size.min(3));
```

Update and save with EOL

- Hawk normally needs to re-read files to notice the changes (indexer)
- We have to update initial.xmi on disk
- Performance hit!

```
1  for (change in Changes!ElementaryChange.all) {
2    change.apply();
3  }
4
5  operation Changes!AssociationCollectionInsertion apply() {
6    if (self.feature.isChangeable) {
7      self.affectedElement.eGet(self.feature).add(self.addedElement);
8    }
9  }
10
11 operation Changes!AssociationPropertyChange apply() {
12   if (self.feature.isChangeable) {
13     self.affectedElement.eSet(self.feature, self.newValue);
14   }
15 }
16
17 operation Changes!AttributePropertyChange apply() {
18   if (self.feature.isChangeable) {
19     self.affectedElement.eSet(self.feature, self.newValue);
20   }
21 }
22
23 operation Changes!CompositionListInsertion apply() {
24   if (self.feature.isChangeable) {
25     self.affectedElement.eGet(self.feature).add(self.index, self.add
26   }
27 }
28
```

Solutions implemented: optimised update

- Initialize, load, initial view: same as before
- Update:
  - Load changeX.xmi, use it to update Neo4j directly
    - Uses a custom "updater" component in Hawk
    - No need to save initial.xmi
  - Update derived attributes incrementally as usual
  - Run original query in EOL

Propagating change events to Neo4j: iterating through them

```
private void applyChangeSequenceToGraph(ModelChangeSet changeSet) throws Exception {
    for (Iterator<EObject> itEObject = EcoreUtil.getAllProperContents(changeSet, false); itEObject.hasNext(); ) {
        EObject eob = itEObject.next();
        if (eob instanceof AssociationCollectionInsertionImpl) {
            applyChangeSequenceToGraph((AssociationCollectionInsertionImpl) eob);
        } else if (eob instanceof AssociationPropertyChangeImpl) {
            applyChangeSequenceToGraph((AssociationPropertyChangeImpl) eob);
        }  else if (eob instanceof AttributePropertyChangeImpl) {
            applyChangeSequenceToGraph((AttributePropertyChangeImpl) eob);
        }  else if (eob instanceof CompositionListInsertionImpl) {
            applyChangeSequenceToGraph((CompositionListInsertionImpl) eob);
        } else if (eob instanceof ElementaryChange) {
            throw new IllegalArgumentException("Unknown elementary change " + eob);
        }
    }
}
```

Propagating change events to Neo4j: using them (watch out for basicGetX)

```
private void applyChangeSequenceToGraph(CompositionListInsertionImpl change) throws Exception {
    addReference((EReference) change.getFeature(), change.basicGetAffectedElement(), change.getAddedElement());
}

private void applyChangeSequenceToGraph(AttributePropertyChangeImpl change) throws Exception {
    IGraphNode node = findNodeByID(change.basicGetAffectedElement());
    node.setProperty(change.getFeature().getName(), change.getNewValue());
}

private void applyChangeSequenceToGraph(AssociationPropertyChangeImpl change) throws Exception {
    final EReference eref = (EReference) change.getFeature();
    clearReferencesFrom(eref, change.basicGetAffectedElement());
    addReference(eref, change.basicGetAffectedElement(), change.basicGetNewValue());
}

private void applyChangeSequenceToGraph(AssociationCollectionInsertionImpl change) throws Exception {
    addReference((EReference) change.getFeature(), change.basicGetAffectedElement(), change.basicGetAddedElement
}
```

Propagating change events to Neo4j: updating nodes

- We never use initial.xmi anymore - we update nodes in the graph directly
- We find the node in the graph by intrinsic ID, using indexed attributes on Post, Comment and User ("id")

```
private void addReference(final EReference eref, final EObject sourceEObject,
        final EObject targetEObject) throws Exception {
    final IGraphDatabase graph = indexer.getGraph();
    final IGraphNode sourceNode = findNodeByID(sourceEObject);
    final IGraphNode targetNode = findNodeByID(targetEObject);

    Map<String, Object> props = new HashMap<>();
    if (eref.isContainment()) {
        props.put(ModelElementNode.EDGE_PROPERTY_CONTAINMENT, "true");
    }
    if (eref.isContainer()) {
        props.put(ModelElementNode.EDGE_PROPERTY_CONTAINER, "true");
    }

    graph.createRelationship(sourceNode, targetNode, eref.getName(), props);
    indexer.getCompositeGraphChangeListener().referenceAddition(commitItem, sourceNode, targetNode, eref.getName()
}
```

Solutions implemented: optimised update + query

- Initialize, load:
  - Almost the same as before
  - No derived attributes used here, though
- Initial view: run original query and store top 3 results
- Update:
  - Register change listeners on the graph
  - Use changeX.xmi to update Neo4j directly again
    - Track which users/comments/posts are changed
  - Rescore impacted elements
  - Merge rescored elements with previous top 3
    - We assume monotonically increasing scores

Updating the top 3 by rescoring updated nodes in the graph (I)

```
@SuppressWarnings("unchecked")
@Override
protected List<List<Object>> runQuery(StandaloneHawk hawk)
        throws IOException, InvalidQueryException, QueryExecutionException {
    if (prevResults == null) {
        prevResults = (List<List<Object>>) hawk.eol(query.getFullQuery());
        hawk.getIndexer().addGraphChangeListener(listener);
    } else {
        // Rescore updated posts
        Operation scoreOp;
        try {
            scoreOp = getRescoreEOLModule(hawk).getOperations().getOperation("score");
        } catch (Exception e) {
            throw new QueryExecutionException(e);
        }
```

Updating the top 3 by rescoring updated nodes in the graph (II)

```
        try (IGraphTransaction tx = hawk.getIndexer().getGraph().beginTransaction()) {
            if (query == Query.Controversial) {
                updateResults(hawk, scoreOp, listener.getUpdatedPostIdentifiers());
            } else {
                updateResults(hawk, scoreOp, listener.getUpdatedCommentIdentifiers());
            }
            tx.success();
        } catch (Exception e) {
            throw new QueryExecutionException(e);
        }
    }

    listener.getUpdatedCommentIdentifiers().clear();
    listener.getUpdatedPostIdentifiers().clear();
    return prevResults;
}
```

Conciseness

- If changes were done directly, Naive can be done with no Java coding at all:
  - Hawk has an Eclipse GUI, we could set up everything manually
  - Only need to write the queries (7 lines of EOL for Q1, 21 lines for Q2)
  - Integrating into benchmark and applying changes required Java coding:
    - EOL update script: 27 lines
    - Other Java code: 770 lines (including comments)
- Incremental update:
  - 400 lines of Java code on top of naive (minus 120 from BatchLauncher)
  - No additional EOL code required
- Incremental update + query:
  - 233 lines of Java code on top of incremental update (minus 120 from BL)
  - Also no additional EOL code required

Correctness

- Kept changing things until the last minute! (2am today)
  - Most of the testing on Q1
  - Almost no testing on Q2 beyond size 1

- Results are as you would expect:
  - Q1 is correct for almost all sizes/iterations from 1 to 64
    - Somehow, two iterations in size 2 fail (need to check)
  - Q2 is correct for sizes 1 and 2, from 4 onwards it is not 100% reliable
    - Sometimes it reports the same elements in a different order
    - Sometimes it reports different elements
    - More debugging needed!

Performance

- Have to hit the disk constantly, unlike other solutions:
  - Hence our order of magnitude slowdown
  - We will consider in-memory Neo4j configurations later

- By mistake, considered some loading times in various steps:
  - Load + save of initial.xmi in Naive
  - Load of changeX.xmi in IncUpdate and IncUpdateQuery

- EOL is interpreted and not compiled
  - Another multiplier on top of having to hit disk
  - Very convenient as a backend-independent query language, though!

Takeaways

- Case was very useful to improve Hawk internally:
  - Lots of little logging improvements (moving away from System.out…)
  - Made a few classes easier to extend by subclassing
  - Improved efficiency of change notifications in local folders
  - Added a new component for monitoring single standalone files
  - Changed Dates to be indexed in ISO 8601 format
  - Added Maven artifact repository to GitHub project
- Learnt a few new bits of EMF black magic:
  - Intrinsic ID maps and DEFER_IDREF_RESOLUTION for initial.xmi loading
  - Differences between EMF *Impl getX() and basicGetX() in proxy resolution
- Got some ideas about:
  - Updating Hawk from EMF change notifications
  - Repackaging query + derived attribute as reusable components
  - Incremental import of XMI files into Hawk

Thank you!