Chapter 1

Gremlin Graph Traversal Language

Contributed by Nathaniel Kremer-Herman

1.1 Background

Gremlin is a distributed graph traversal language. It is used to traverse large graphs which are distributed across different nodes in a cluster, cloud, or grid. Gremlin interoperates with many different distributed graph representations, meaning it can be used to query large graphs on different distributed architectures.

Gremlin is a project by the Apache Software Foundation as part of its Apache Tinkerpop graph computing framework [7]. The Tinkerpop graph computing framework was started to address computational needs for property graphs. A property graph is a way to define a graph such that each vertex and each edge may have an arbitrary number of key-value pairs called properties associated with it. Tinkerpop, and by extension Gremlin, is open-source and vender-agnostic. It is designed as a general-purpose property graph computing framework of which Gremlin is its graph traversal and querying tool. The Gremlin console, Gremlin server, and Tinkerpop project source code are all available at [7].

1.2 Expressing Graphs

Since Gremlin is not a language used to express graphs, it supports many types of graph representations which it can in turn traverse. It supports two categories of traversals: online transactional processes (OLTP) and online analytics processes (OLAP) [9]. OLTP is characterized by real-time database querying of data, and the graph is represented by relationships between different elements in the database. OLAP represents batch processing systems. Each of these batch processing systems has its own method of expressing graphs which Gremlin supports for its traversals. Gremlin supports many popular OLTP graph databases such as OrientDB [8], Titan [1], and Blazegraph [2]. Gremlin also supports many OLAP graph processors like Hadoop [5], Giraph [4], and Spark [6]. More supported OLTP and OLAP systems can be found in the Gremlin documentation [9].

1.3 Syntax

Gremlin queries are constructed using a functional, data-flow syntax. These queries are then executed in the special-purpose Gremlin Console [7]. A Gremlin query conditionally traverses a

graph by examining the properties of vertices and edges as discussed in Section 1.1. As an example, retrieving a list of people which Jane Doe knows is done with the following Gremlin query:

```
g.V().has("name","Jane Doe").out("knows").values("name")
```

In this query, `g` represents the graph, and `v` refers to the current vertex (as Gremlin traverses the graph as explained in Section 1.5). The `has` function matches a property key to a propert value (in this case the key is *name* and the value is *Jane Doe*). The `out` function takes in a property key (i.e. *knows*) and only traverses outbound edges which have a property with that key. Finally, the `values` function produces a list of values referring the the given key (i.e. *name*).

As a more complex example, we can see how a Gremlin query can be used to find the top ten ranked people in a social network by PageRank:

```
g.V().hasLabel("person").pageRank().by("friendRank").by(outE("knows")).
order().by("friendRank",desc).limit(10)
```

The syntax makes it easy to comprehend what the query will return since each function calls builds off the previous much like Scala or JavaScript callbacks. Working backward, the query returns the first ten items of a descending order list by friend rank which is produced by the outbound edge property key *knows* on a PageRank calculation of the friend rank of every vertex which has the label *person*. Gremlin queries entered into the Gremlin Console can be arbitrarily complex, but these two examples demonstrate the expressiveness of the syntax. In addition to this syntax, Gremlin has bindings in other programming languages allowing for Gremlin queries to be defined in-line in scripts. Languages supported include Ruby, Python, Groovy, JavaScript, and PHP [9].

1.4 Key Graph Primitives

Discuss here what are the key graph primitives supported by the paradigm.

Since Gremlin defines queries and traversals of a graph and not the graph representation itself, it does not make use of graph primitives *per se*. Rather, it makes assumptions about the primitives the OLTP and OLAP systems support since the goal of Gremlin (and the Apache Tinkerpop project as a whole) is to be a vendor-agnostic graph computing framework. There are really only two primitives Gremlin assumes exist. Those are vertices and edges. On each vertex and edge, there can be arbitrarily many key-value pair properties as discussed in Section 1.1. There must be *at least* one property on every edge and vertex. This means Gremlin traverses and queries property graphs exclusively.

Because of the way property graphs are represented, there may be arbitrarily many edges between two vertices. Taking inspiration from the examples in Section 1.3, two vertices A and B may have an edge connecting them with the property *knows* as well as an edge between them with the property *friend*. The edges are assumed to be directional, so each vertex would need to have its own *knows* and *friend* edges pointing to the other. It is also possible, since an edge can have multiple properties, that both *knows* and *friend* could be stored on a single edge. Gremlin allows for this to occur, but some of the systems it supports may not have that functionality.

1.5 Execution Model

There are many layers to the execution of a Gremlin query. At the most fundamental layer, the user interacts with a special-purpose console called the Gremlin Console. Through this console, a user writes their Gremlin queries as described in Section 1.3. The Gremlin Console then sends the query to a Gremlin Server for processing. The Gremlin Server is a front-end for a cluster, cloud, or

grid of distributed machines which each process the query across the vertices of the graph stored at that machine.

On each machine is a Gremlin Traversal Machine (GTM). The GTM is a special-purpose Java Virtual Machine (JVM). Following the *write once, run anywhere* philosophy of the Java programming language, each GTM instantiates and is able to parse an incoming Gremlin query without any added configuration by an end-user. The GTM is aware of what set of vertices and edges (both inbound and outbound) are stored at its compute node.

When a Gremlin query is submitted from the Gremlin Console to a Gremlin Server, a process called the Gremlin Traverser is instantiated. The Gremlin Traverser is the process in charge of executing the query on a cluster, cloud, or grid. It starts at some arbitrary vertex of the graph which will be located on a known, single machine. The Gremlin Traverser then performs the query as far as it can on the current machine. When a transition from one vertex to another on a different machine is required, the state of the Gremlin Traverser is transferred to the GTM on the next machine over the network. Figure 1.1 describes in detail the architecture of executing a Gremlin query.

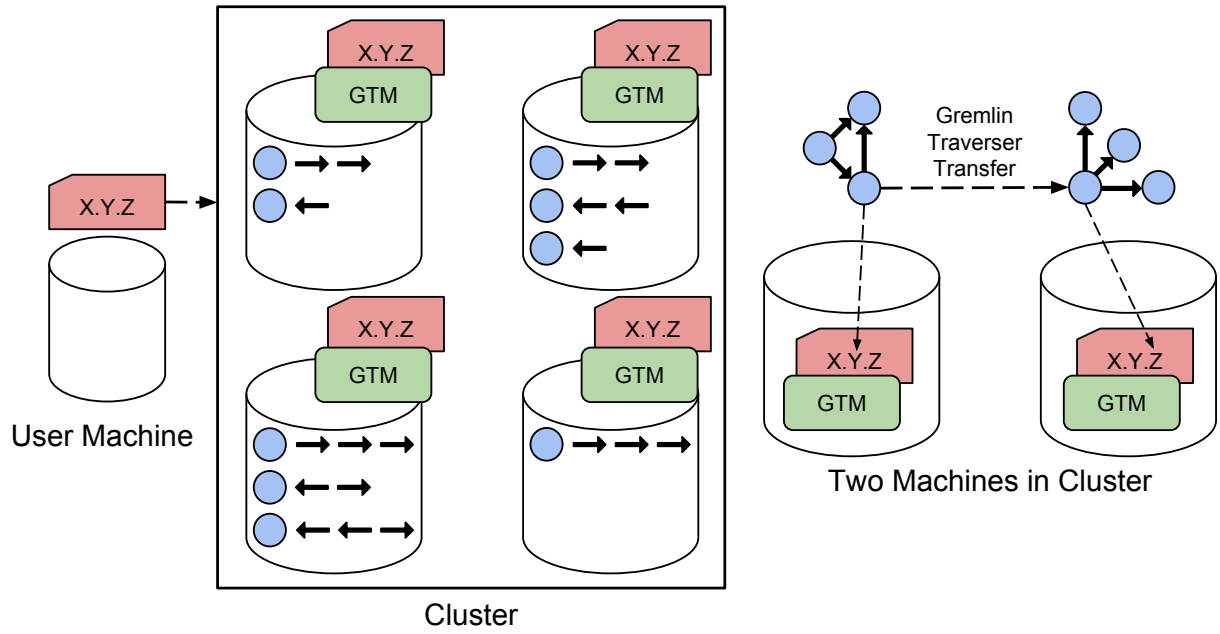

Figure 1.1: Gremlin Execution Architecture.

A graph query `x.y.z` is defined at the user machine via the Gremlin Console which is submitted to a cluster by a Gremlin Server (not shown). In the cluster, each machine runs a Gremlin Traversal Machine and obtains a copy of the query from the Gremlin Server. Each machine also has a portion of the entire graph stored (either in memory or on disk) as a set of vertices and associated edges. When executing a query, a Gremlin Traverser process enters a GTM, performs its traversal on relevant vertices, then determines via edge properties which machine it must reach to continue its traversal. A small example showing two machines in this cluster is shown.

1.6 Examples

At the time of this report, Gremlin has not been referenced in peer-reviewed work demonstrating its performance or its use in any specific application. Neither the Gremlin website nor the Apache Tinkerpop website provide performance reports or example applications besides the syntax examples provided in their documentation. For the sake of completeness, we provide an example query for a somewhat realistic though small-scale use case. In this example, we want to find which song writers wrote songs that were sung by Jerry Garcia and performed by the Grateful Dead more than 300 times [3].

```
g.V().match(as("song").out("sungBy").as("artist"),
as("song").out("writtenBy").as("writer"),as("artist").
has("name","Garcia"),where(as("song").values("performances").
is(gt(300)))).select("song","writer").by("name")
```

The Gremlin Console produces the following output when executing this query:

```
gremlin> :> @query
==>[songName:BERTHA, writerName:Hunter]
==>[songName:TENNESSEE JED, writerName:Hunter]
==>[songName:BROWN EYED WOMEN, writerName:Hunter]
==>[songName:CHINA CAT SUNFLOWER, writerName:Hunter]
==>[songName:CASEY JONES, writerName:Hunter]
==>[songName:BLACK PETER, writerName:Hunter]
==>[songName:RAMBLE ON ROSE, writerName:Hunter]
==>[songName:WHARF RAT, writerName:Hunter]
==>[songName:LADY WITH A FAN, writerName:Hunter]
==>[songName:HE'S GONE, writerName:Hunter]
==>[songName:LOSER, writerName:Hunter]
==>[songName:DEAL, writerName:Hunter]
==>[songName:SUGAREE, writerName:Hunter]
==>[songName:DON'T EASE ME IN, writerName:Traditional]
==>[songName:UNCLE JOHNS BAND, writerName:Hunter]
==>[songName:SCARLET BEGONIAS, writerName:Hunter]
==>[songName:EYES OF THE WORLD, writerName:Hunter]
==>[songName:US BLUES, writerName:Hunter]
==>[songName:TERRAPIN STATION, writerName:Hunter]
==>[songName:STELLA BLUE, writerName:Hunter]
gremlin>
```

1.7 Conclusion

Gremlin is a useful vendor-agnostic tool for large graph traversal and querying. It is capable of performing its graph traversal in a distributed manner through the Gremlin Traversal Machine which can coordinate the distributed traversal over the network. Its inclusion in the Apache Tinkerpop graph computing framework means it can support many different online transactional processes (OLTP) and online analytics processes (OLAP) systems. Gremlin's syntax provides a functional programming interface reminiscent of Scala and JavaScript callbacks. This will feel very familiar to Scala programmers, and the fact that Gremlin has Scala support seems to point to this lineage.

Overall, Gremlin provides an expressive, straightforward way to traverse and query distributed graphs.

Bibliography

[1] Think Aurelius. Titan: Distributed graph database, 2018.

[2] Blazegraph. Blazegraph: Graph database & application, 2018.

[3] Datastax. The benefits of the gremlin graph traversal machine, 2018.

[4] Apache Software Foundation. Apache giraph, 2018.

[5] Apache Software Foundation. Apache hadoop, 2018.

[6] Apache Software Foundation. Apache spark, 2018.

[7] Apache Software Foundation. Apache tinkerpop, 2018.

[8] OrientDB. Orientdb: Graph database — multi-model database, 2018.

[9] Apache Tinkerpop. The gremlin graph traversal machine and language, 2018.