Aggregators

What are aggregators?

Aggregators provide a way to perform global computation in your application. You can use them for example to check whether a global condition is satisfied or to keep some statistics.

During a superstep vertices provide values to aggregators, these values get aggregated by the system and the results become available to all vertices in the following superstep. Providing a value to aggregator is done by calling

```
aggregate(aggregatorName, aggregatedValue)
```

and to get the value aggregated during previous superstep you should call:

```
getAggregatedValue(aggregatorName)
```

Operation which aggregator is performing should be commutative and associative, since there is no guarantee of the order in which the aggregations will be performed.

Regular vs persistent aggregator

Aggregators come in two flavors: regular and persistent aggregators. The value of a regular aggregator will get reset to the initial value in each superstep, whereas the value of persistent aggregator will live through the application.
As an example, consider having SumAggregator, and each vertex adds 1 to it during compute. If this is regular aggregator, you'll be able to read the number of vertices in the previous superstep from it. If this is persistent aggregator, it will hold the sum of the number of vertices in each of the previous supersteps.

Registering aggregators

Before using any aggregator, you ***MUST register it on the master***. You can do this by extending (and setting) MasterCompute class, and calling:

```
registerAggregator(aggregatorName, aggregatorClass)
```

or

```
registerPersistentAggregator(aggregatorName, aggregatorClass)
```

depending on what kind of aggregator do you want to have. You can register aggregators either in MasterCompute.initialize() - in that case registered aggregator will be available through whole application, or you can do it in MasterCompute.compute() - aggregator will then be available in current and each of the following supersteps.

Aggregators and MasterCompute

The first thing which gets executed in a superstep is MasterCompute.compute(). In this method you are able to read aggregated values from previous superstep by calling

```
getAggregatedValue(aggregatorName)
```

and you are able to change the value of the aggregator by calling

```
setAggregatedValue(aggregatorName, aggregatedValue)
```

Implementations

In the package org.apache.giraph.aggregators you can find many common aggregators already implemented. If you need to create your own, you can extend DefaultAggregator or Aggregator class.

Advanced options

If you are using multiple threads for computation (giraph.numComputeThreads), you should consider turning on giraph.useThreadLocalAggregators option. The downside is that it will use more memory - you'll have several copies of each of the aggregators. So if you have a lot of aggregators, or aggregated values are very large objects, this option could be bad. But otherwise, it might speed up your application, since it will remove the need to perform synchronization when aggregating values.