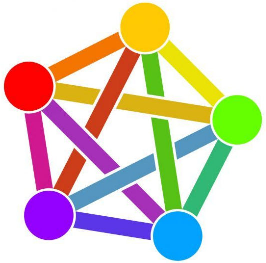
Pregel

Google's Large Scale Graph Processing System

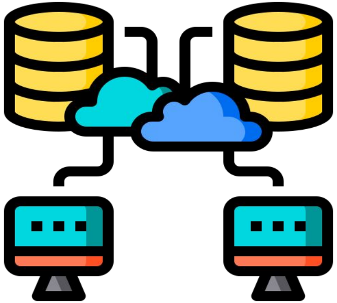
COL733 PROJECT

- Geetansh Juneja (2020CS50649)
- Hemank Bajaj (2020CS10349)
- Shivam Verma (2020CS50442)

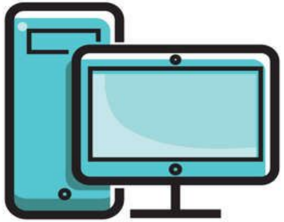
Motivation



Need of parallel computation for Large Scale Graphs



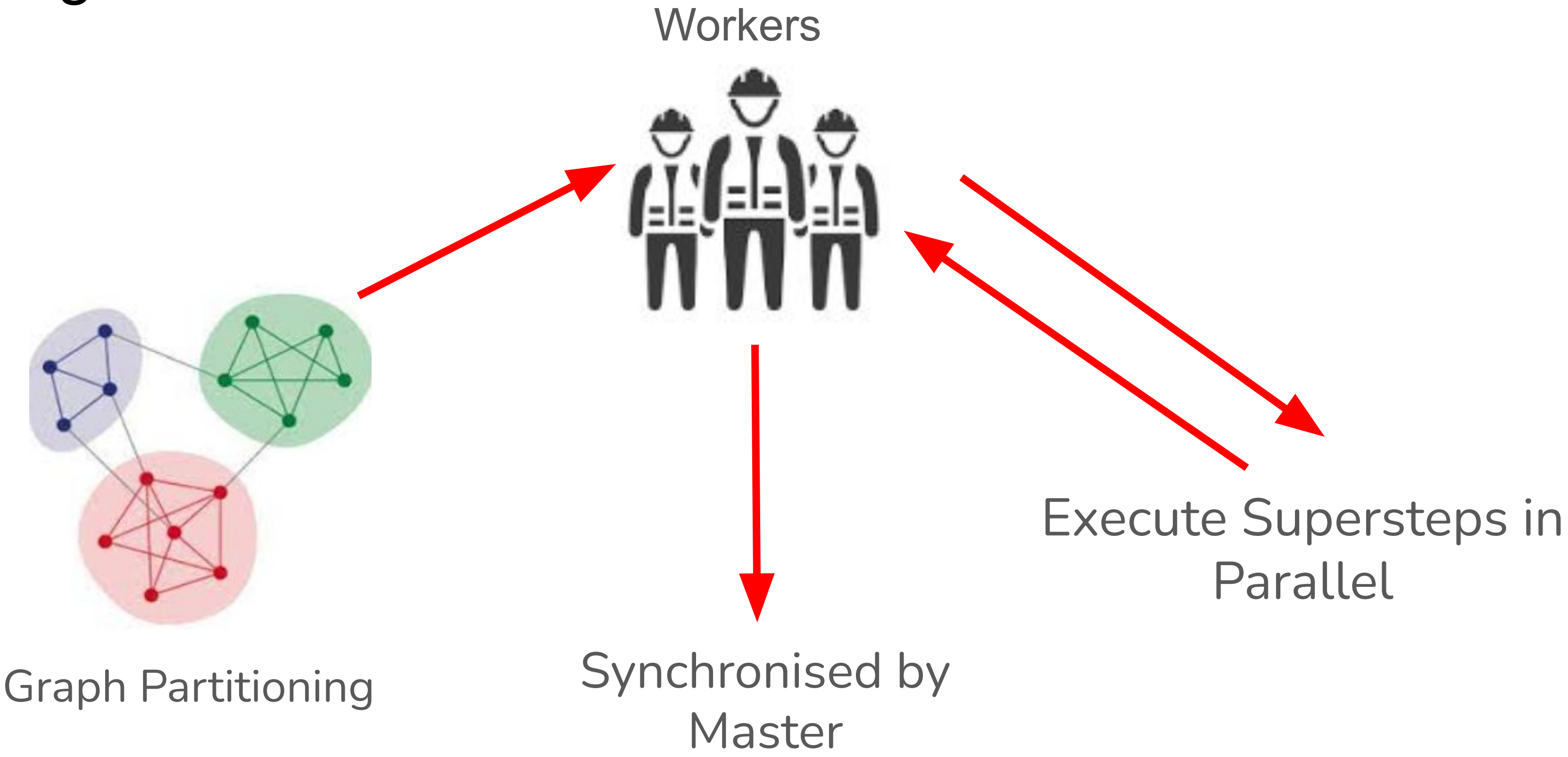
The graph data may be stored on different machines in a data server



We need an interface to perform computations on large scale distributed graphs

PREGEL

Pregel : Architecture



Workers

Graph Partitioning

Synchronised by Master

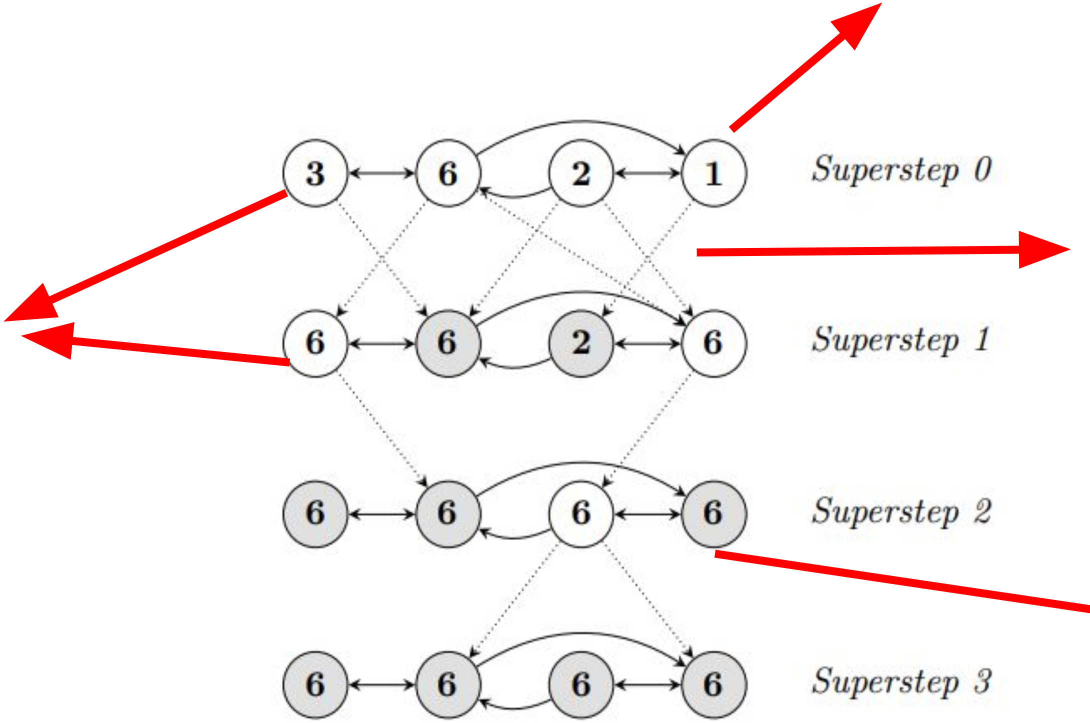
Execute Supersteps in Parallel

Pregel : Key Features

1. Input is a directed graph, where each vertex has unique ID.
2. Vertices have modifiable, user-defined values, edges, and list of incoming and outgoing messages.
3. Pregel computation operates in a series of iterations called "supersteps".
4. During each superstep, the worker machine calls a user defined compute() function on each vertex of its partition.
5. Compute() function: Dictates behavior of a vertex in a superstep by reading messages, sending new messages, and modifying the vertex state and its outgoing edges.
6. A vertex may also vote to halt in compute() function.
7. Termination: When all vertices are inactive and no messages are in transit.

Supersteps: Heart of Pregel

Call compute() function on each vertex



Supersteps

In compute():
- Update its value
- Send messages to superstep S + 1
- Read messages sent in superstep S - 1
- Halt

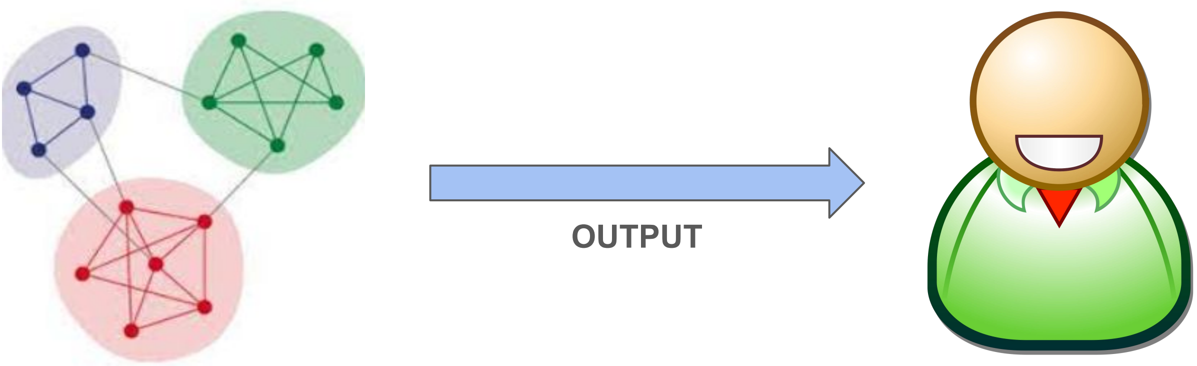
Some Vertices Halt!

Computation is complete once all the vertices halt.

Pregel : Termination and Output

When all workers have finished computation on their partition, the master instructs each worker to save its portion of graph.

The user may then read the final values of each vertex as output.

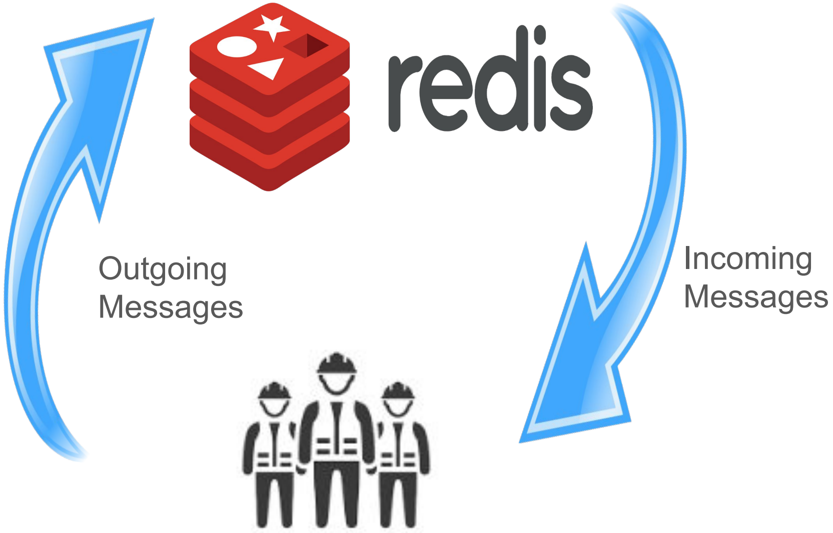


OUTPUT

Postgel : Design Choices

Single Machine Implementation
Processes simulate worker machines

Use of Redis for Message Transfers



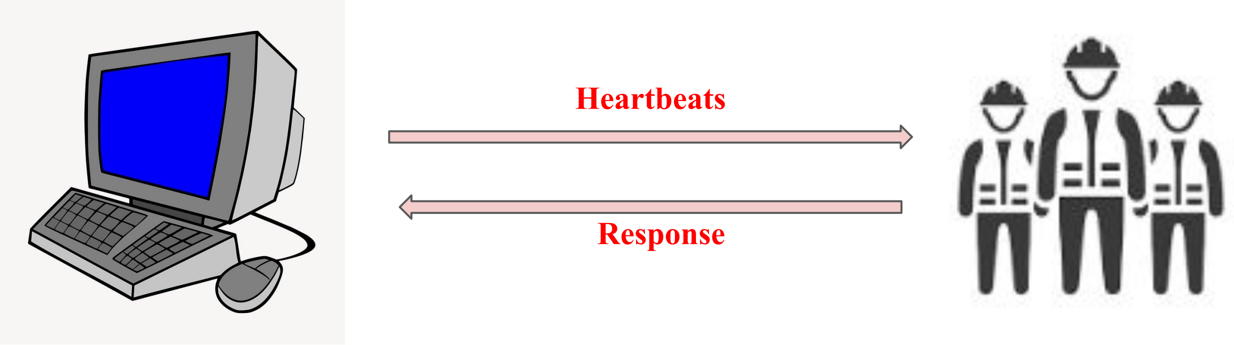
Postgel : Implementation Details

- **Postgel Master**
  - Responsible for graph partitioning. Hashing of node ID
  - Establishes connection to Redis
- **Vertex**
  - Incoming and Outgoing Messages
  - Users override the update() function for their use-cases.
- **Workers (Python Processes)**
  - Communicate with redis to send and receive messages
  - Execute supersteps on the partition assigned
- **Message Passing**
  - Message passing across supersteps done through Redis.
  - Aggregator implemented to improve message passing

Postgel : Fault Tolerance

Our design is also Fault Tolerant against worker failures.

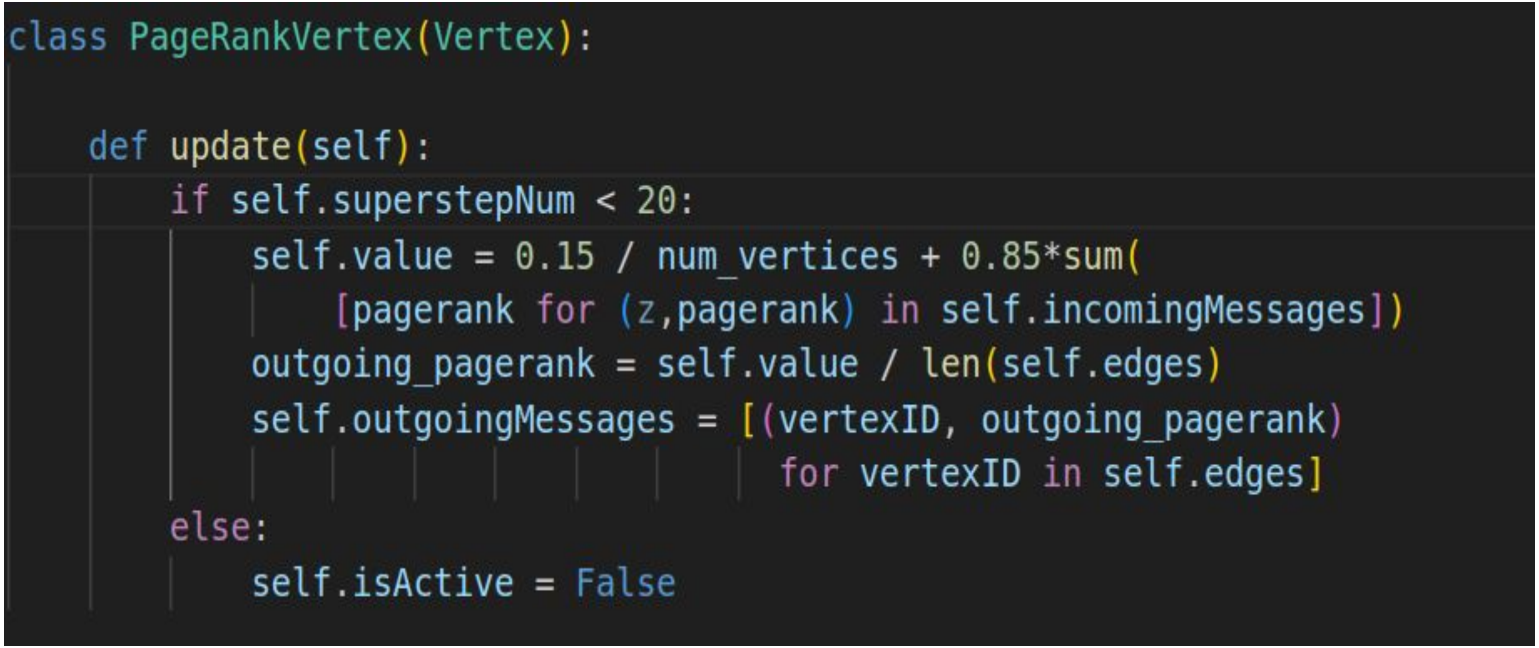
- Each worker checkpoints the state of its partition after every 2 supersteps.
- Master sends heartbeat to the worker after every 2 seconds.
- Worker is marked failed if it doesn’t respond to heartbeat.
- Master redistributes the partition from the last checkpointed superstep and restarts the alive workers.



Heartbeats

Response

Using Postgel to compute PageRank

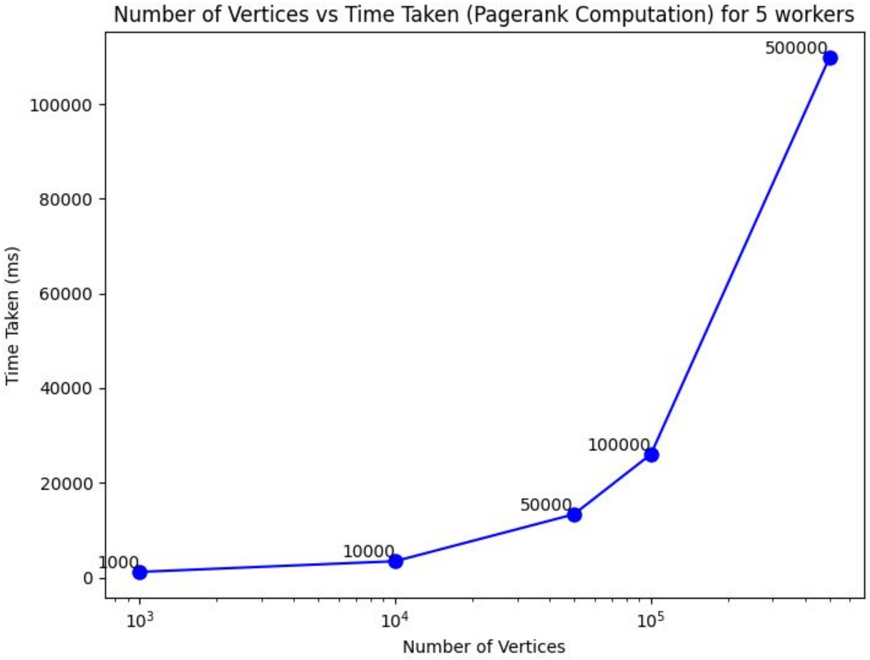


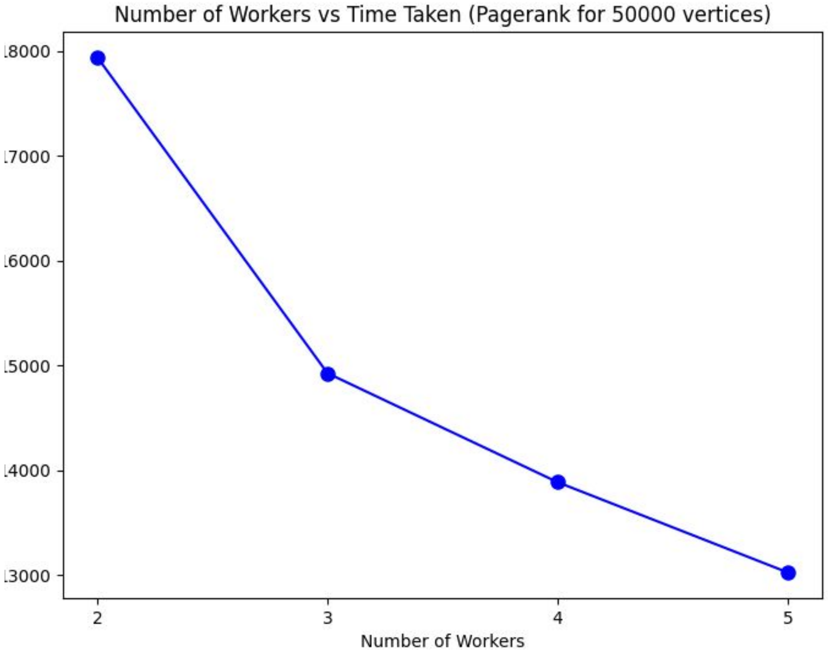
```python
class PageRankVertex(Vertex):

    def update(self):
        if self.superstepNum < 20:
            self.value = 0.15 / num_vertices + 0.85*sum(
                [pagerank for (z,pagerank) in self.incomingMessages])
            outgoing_pagerank = self.value / len(self.edges)
            self.outgoingMessages = [(vertexID, outgoing_pagerank)
                                      for vertexID in self.edges]
        else:
            self.isActive = False
```

In this example, the value of each vertex starts converging after certain number of supersteps. Here we run it for 20 iterations.

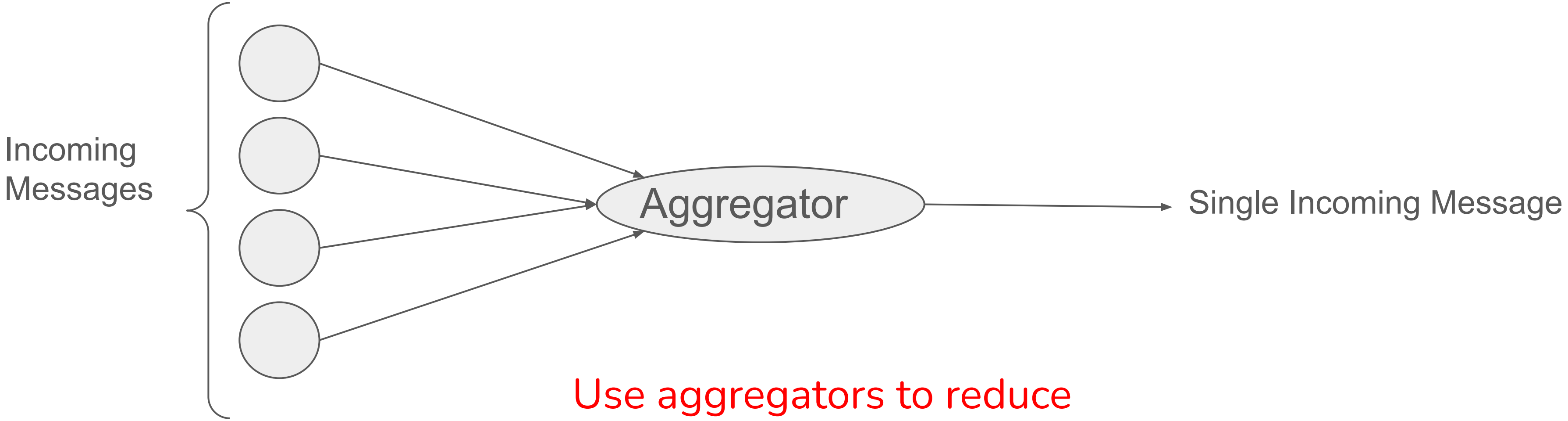
Page Rank Example : Plots





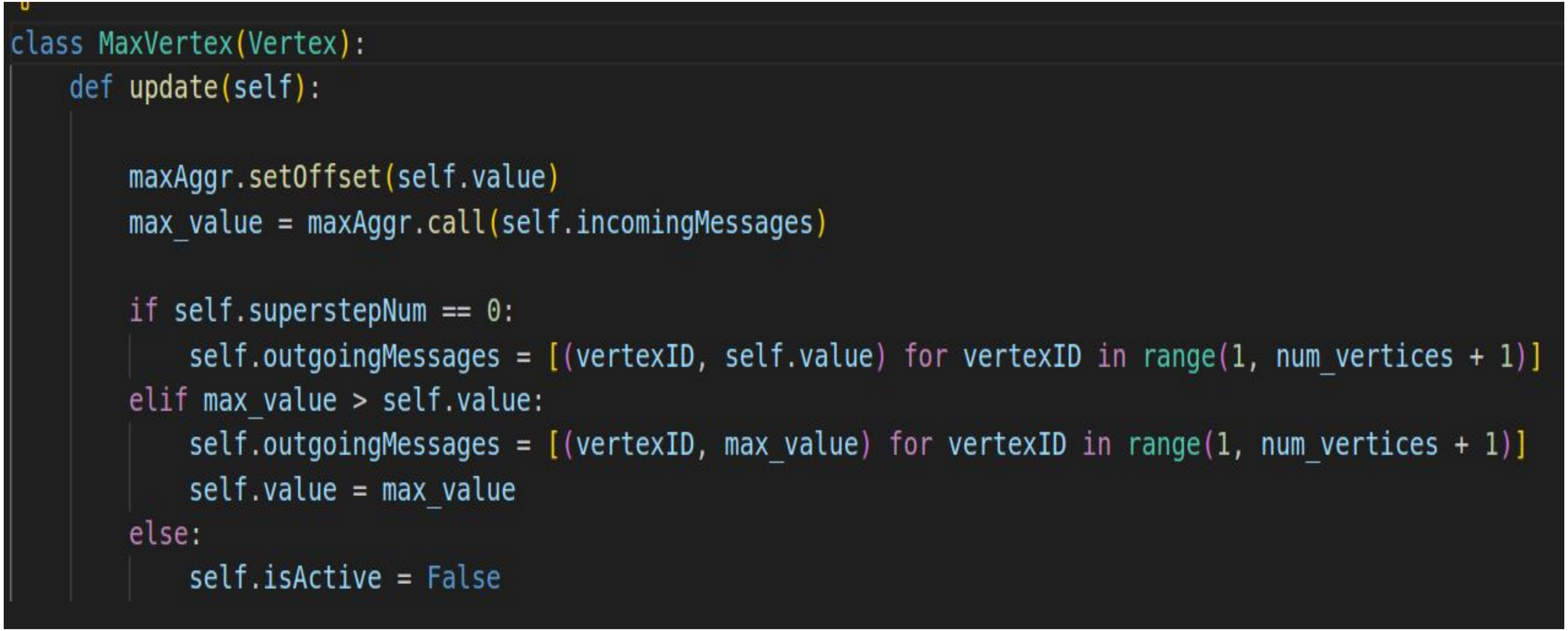
Aggregators

Functions like max, min, sum used often



Use aggregators to reduce communications with Redis!

Max Node Computation : Aggregator Example



```
class MaxVertex(Vertex):
    def update(self):

        maxAggr.setOffset(self.value)
        max_value = maxAggr.call(self.incomingMessages)

        if self.superstepNum == 0:
            self.outgoingMessages = [(vertexID, self.value) for vertexID in range(1, num_vertices + 1)]
        elif max_value > self.value:
            self.outgoingMessages = [(vertexID, max_value) for vertexID in range(1, num_vertices + 1)]
            self.value = max_value
        else:
            self.isActive = False
```

Self-Introspection : Pros and Cons of our Design

- PROS
  - General Purpose system. User specified functions
  - Aggregators improve efficiency for specific use-cases
  - System is uninterrupted despite of worker failures
- CONS
  - Redis communications can be an overhead
  - Writing graph computations in Pregel can be challenging
  - Cannot run more than 8 parallel workers in Single Machine Design

Pregel VS MapReduce: Pregel keeps vertices and edges on the machine that performs computation, and uses network transfers only for messages. MapReduce, however, is essentially functional, so expressing a graph algorithm as a chained MapReduce requires passing the entire state of the graph from one stage to the next - in general requiring much more communication and associated serialization overhead.

Thank You

Feel free to ask questions!

Our Implementation and detailed report can be found here