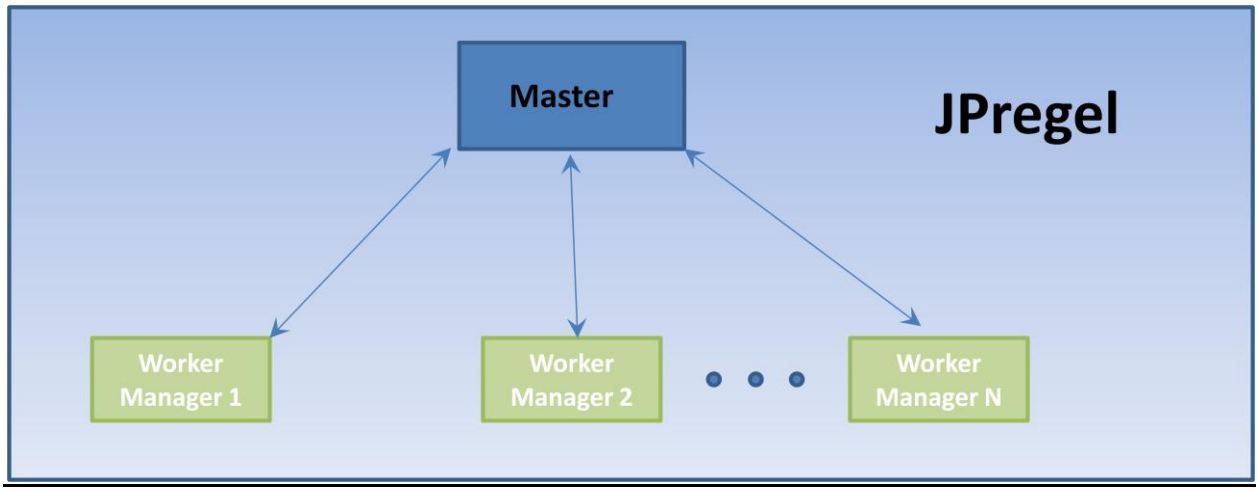
JPREGEL - DESIGN DOCUMENT and TECHNICAL CHALLENGES

Pregel is a system for large scale graph processing developed by Google. The aim of this project to implement Pregel in Java using RMI.

The design of Jpregel is based on a vertex centric approach where each vertex in the graph executes a user defined function. The system composes these actions to lift the results to a large data set. The design is based on the Bulk synchronous parallel model which works as given below

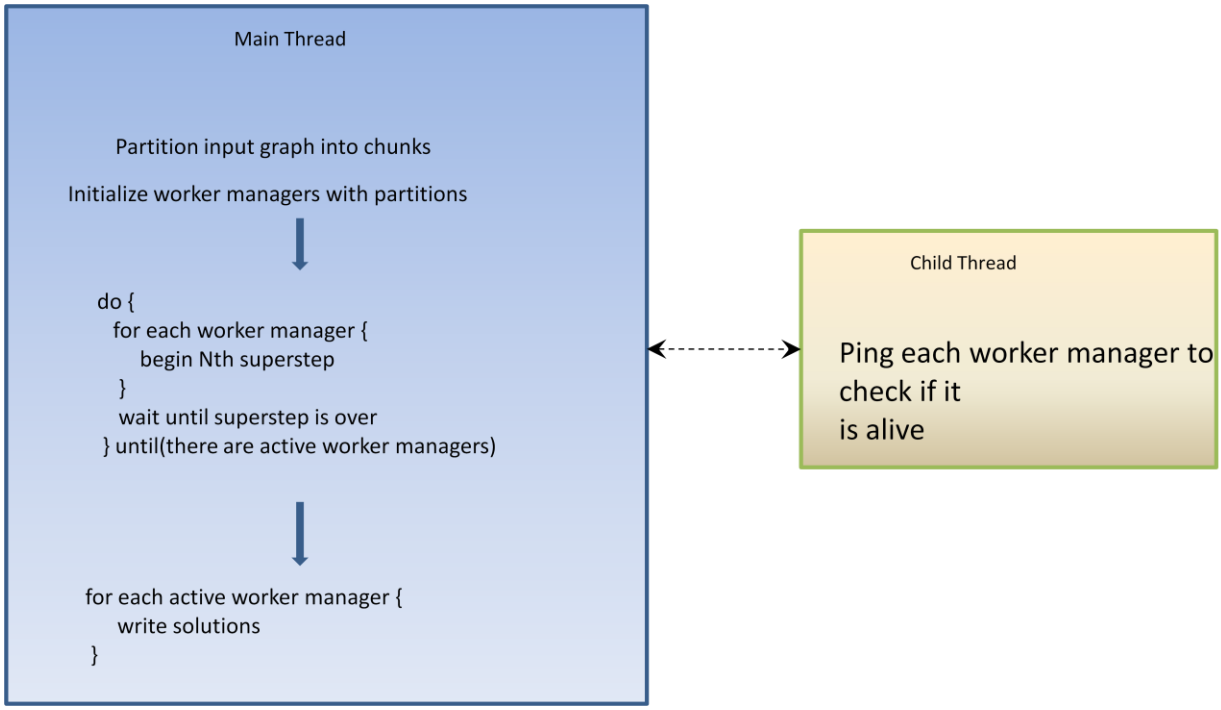
1. Local computation – A user defined function is executed at each vertex
2. Communication – Vertices communicate with each other
3. Barrier Synchronization – The system waits for all the vertices to complete one iteration of computation

System Architecture:



The overall architecture of JPregel follows a master – worker model where a master controls a number of worker managers. Each worker manager represents a machine in a cluster, each of which will run a number of worker threads.

Architecture of Master:



The master has 2 threads:

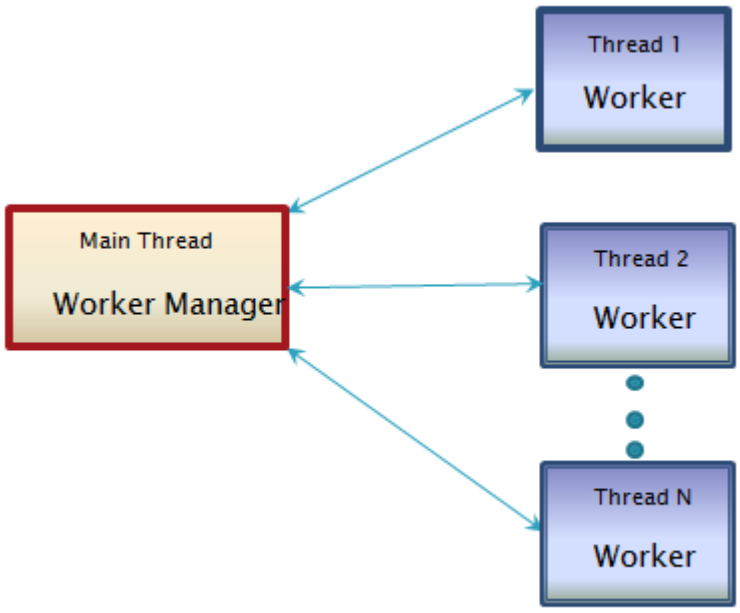
1. Main thread – The main thread does the following functions:

- Partitions the input graph into an optimum number of chunks depending on the number of machines in the cluster and the number of worker threads each machine is capable of running.
- Initializes the worker managers by dividing these partitions among them.
- Maintains a list of worker managers which are active.
- Executes supersteps until there is atleast one worker manager which is in active state
- After all the worker managers have become inactive the master directs all the worker managers to write solutions for all the vertices that are part of the partitions it is handling.

2. Child Thread- The function of the child thread is as follows

- Poll all the worker managers to check if they are alive.
- If anyone of them is not alive then stop the master thread from executing supersteps further.
- Initiate recovery operations.
- Signal all the active worker managers to stop execution.
- Redivide the partitions among the active set of worker managers
- Begin the execution again from the previous checkpoint.

Architecture of Worker Manager:



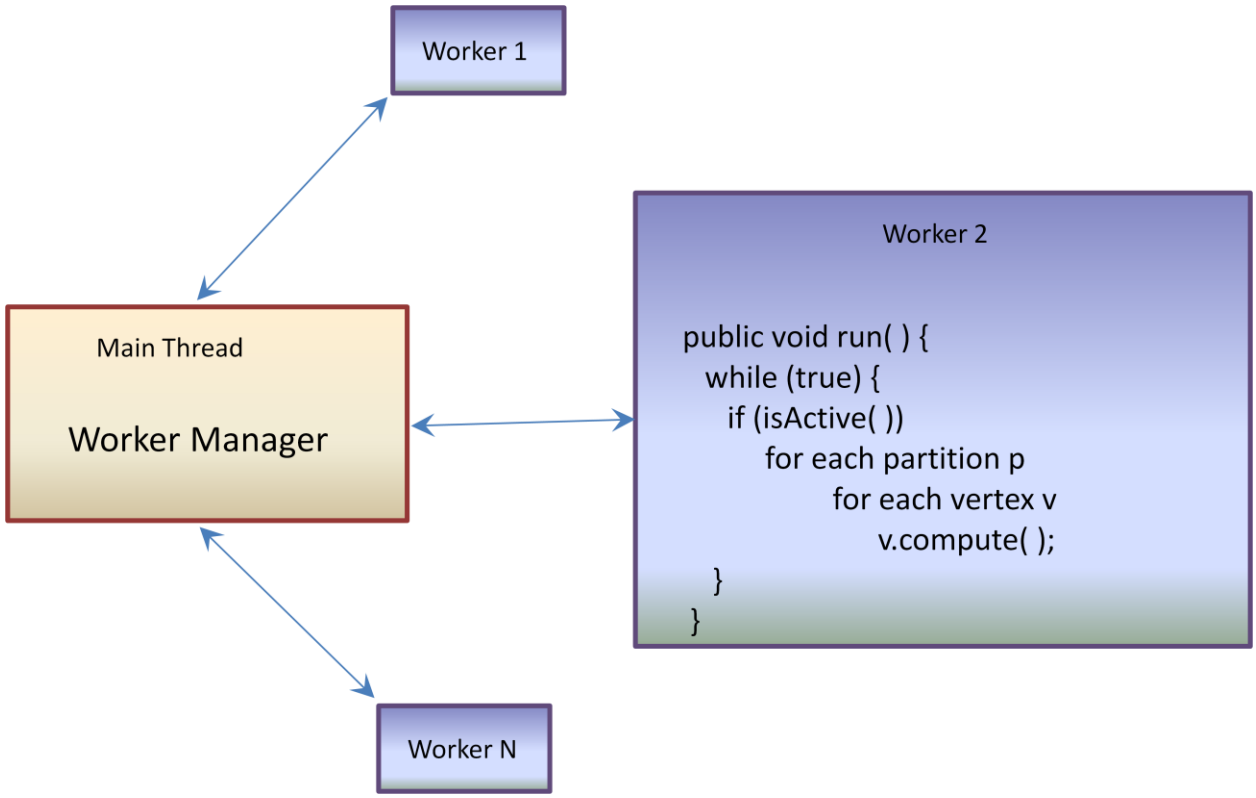
Each worker manager corresponds to one compute node in the cluster. It handles a number of worker threads. It acts as an interface between the master and the workers.

The functions of the worker manager are

- Initialize all the worker threads by distributing the partitions it has received from the master to each worker that it handles.
- Communicate messages to start and end supersteps, received from the master to all workers.
- Buffer all messages received from all other vertices at the end of a superstep and distribute them to the intended vertex queue at the beginning of every super step.

Architecture of the Worker:

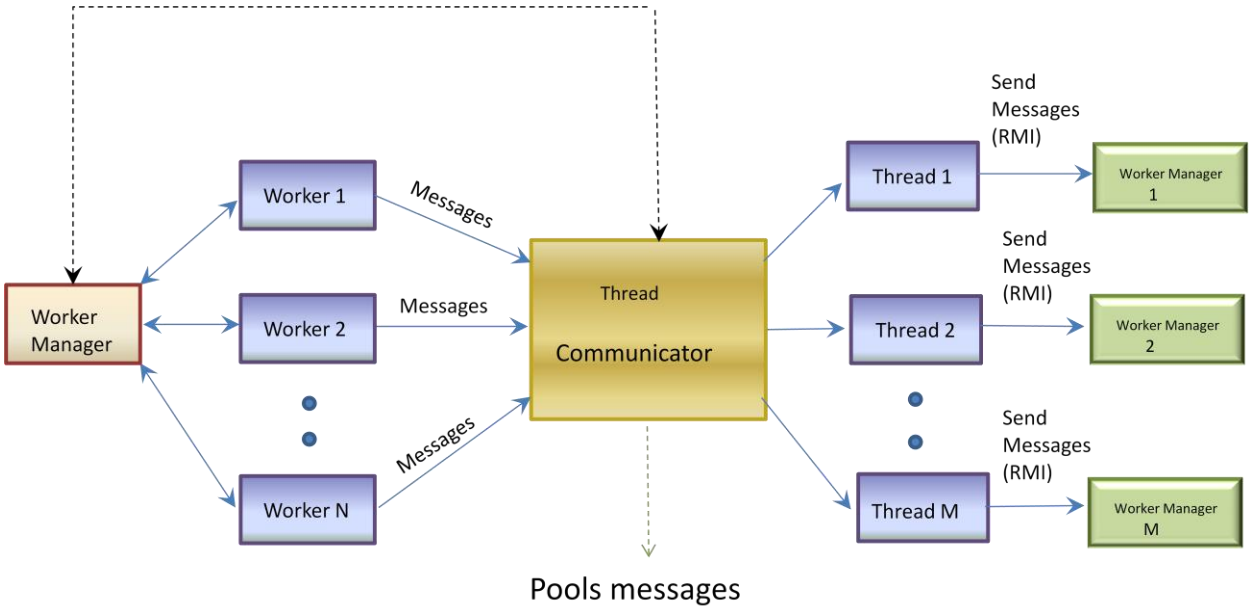
Each worker executes a user defined function which is the compute method provided by the application programmer shown in the diagram below.



Architecture of the Communicator:

All the workers after completion of a superstep send all the messages that they have to send to other vertices the communicator. The communicator looks up the address of the worker manager to whom the messages have to be sent and puts together all the messages that have to be sent to the same destination worker manager.

This method of putting together all messages intended for the same destination worker manager reduces the number of RMI calls.



Fault Tolerance:

To recover from the failure of machines in the cluster checkpointing procedures are implemented.

The state(the value of the vertex, list of its outgoing edges, the cost of all outgoing edges and the list of messages in the queue of every vertex) of all the vertices in every compute node is periodically written to the disk.

The child thread in the master polls all the machines in the cluster to check if they are alive.

Whenever there is a fault detected

- The child thread of the master stops the master thread from executing furthersteps.
- It sends a stop message to all the active worker managers to stop them from continuing execution.
- It redistributes the partitions among the list of active worker managers.
- It sends a restore message to all the worker managers.
- The worker managers read the state of the newly assigned partitions from the disk and reload their states.
- The child thread then sends a message to all the worker managers to restart from the superstep where the previous checkpoint was done.

TECHNICAL CHALLENGES

Graph Partitioning:

Static Partitioning – The division of vertices into partitions is done only once.

Static partitioning of the graph can be done in 2 ways

1. Divide the graph such that each worker thread receives one partition. The size of the partitions in this case becomes huge. In case of a fault, reassigning the partitions from the failed machine will burden one of the machines which would get the extra partition.
2. Divide the graph into small chunks based on the number of workers and number of threads each worker has. In case of failure this method causes an even load distribution to the available machines. But reassigning partitions after a failure here is challenging.

We implement the second method of partitioning in order to achieve load balancing.

Combiners:

Every machine will have multiple vertices which are executing the user defined functions. At the end of a superstep there will be many vertices which would send messages to the same destination vertex. Sending each of these messages independently will cause an RMI overhead.

So we implement combiners which combine all the messages intended for the same destination vertex into a single message. So there is only one RMI call per destination vertex.

Communicator:

- Machines receiving the messages should not block on receiving messages. The communicator will write the message into the worker manager queue of the destination and return immediately.
- Remote object Lookup to find the destination machines to send messages should be done only once before the first superstep and once after a failure.
- There are multiple threads for sending messages to different destination worker managers in parallel.