**Basic Challenges**

1. How can a fault-tolerant, reliable MapReduce environment be implemented on a P2P system?
2. How can a DHT be incorporated into the execution of MapReduce jobs?

**Communication Challenges**

1. It cannot be guaranteed that a broadcast message actually arrives, even if a task completed correctly.
2. Handling failed executions does not add any benefit but is only maintenance effort besides the fact that failed executions are very difficult to recognise and handle (what if the “failed task” broadcast never arrives?).
3. There are additional messages on the network that do not contribute to the execution.
4. How can nodes profit from completed tasks and/or procedures by other nodes without relying on their successful execution?
5. What is the minimal number and types of messages to send to avoid unnecessary and possibly very complex message handling?
   1. What *maintenance* messages (node joined, node disconnected, node crashed) are needed?
   2. What actual *execution* messages are needed (completed, finished, failed task, procedure, job)?
   3. How can it be determined if a job is still executed or not)?

**Data Storage Challenges**

1. How can different nodes emit key-value pairs to the same DHT without collisions? As add calls will append and not override values to keys as it would be in the case of put, there will most likely be too many values associated with one task output key as soon as two executors execute the same task, leading to wrong output data.
2. What happens with data of a calculation node that collapses mid-execution?
   1. The data produced by that node should not be used as it cannot be guaranteed to be finished.
   2. But how can it be accomplished that independent task results can be used by other calculation nodes this calculation node’s collapse?
      1. Keys are not aggregated for the complete procedure but only for individual tasks. If the next procedure wants to retrieve all keys and values for those keys, it would have to first retrieve all the individual task keys for each executor task domain.
      2. Even more of a challenge is the fact that an execution node would have to send around all its executor task domains that lead the procedure to its completion. If we imagine a Wordcount Mapper that produced 100 000 tokens, 100 000 executor task domains would have to be sent around.
3. Memory (RAM) Challenges: how to avoid overloading RAM of one execution node with too large data sets to produce?