De-centralized Peer-to-Peer MapReduce System

CS647 Final Presentation, Spring 2009

Omar Badran
Jordan Osecki
Bill Shaya

Overview

- Introduction
- Background
- Approach
- Evaluation
  - Demo
- Related Work
- Conclusions
- References

Introduction

- Typical MapReduce Systems are comprised of a group of servers housed at a single location
- Characteristics:
  - Jobs are run on the few servers at the location
  - MapReduce systems typically have only one “master” and several “worker” nodes that perform the map and reduce functionality
  - The “master” coordinates the MapReduce operation while the workers actually perform parts of the job

Introduction

- Pitfalls of the "typical" MapReduce:
  - All jobs that are run are decided within that server room
  - The jobs can only utilize resources available there
  - The whole system is dependent on the single master for coordination
  - The job is dependent on all of the workers doing their job
  - It is difficult to switch the role of a node in the process even if new roles would dramatically increase the efficiency of that job
  - It is not very good at reacting to failures, which will most likely need human intervention to resolve them

Introduction

- Proposal: the development of a simulated MapReduce system over a de-centralized Peer-to-Peer (P2P) network
- Benefits:
  - This system will have the ability to have access to numerous servers that are connected to the Internet for both extra computational ability and job submission
  - The de-centralized approach will allow any peer node to act as the master
  - There will no longer be a dependency on any one master or any workers, as new masters can be appointed from the general node mass and new workers can be found to replace any that may have been disconnected

Introduction

- Benefits (continued):
  - No workers will be overburdened, as there will be enough to split up the task into the ideal chunks necessary, choosing the ideal workers for the job
  - This system will employ two self-* innovations in order to further improve the MapReduce experience.
  - All of these changes will make for a more open, efficient, utilized, robust, and reliable system overall than the traditional MapReduce system

Background

- MapReduce systems are used for very large jobs that in some way have tasks which can be performed independently of each other
- A simple example of this is counting the number of words in a file or counting occurrences of a certain word or other trait in a file. This is purely a parallelizable job with each portion independent because no results depend on any others.

Background

- Although this is a trivial job, there are much more important and complicated jobs in all fields of science which can be split up in chunks and have the results calculated in parallel. This saves a lot of time and resources
- For example, if a job is 100% parallelizable and there are 10 workers, it can do it in about 10% of the time as compared to one worker doing the job

Background

- Scenario:
  - MapReduce systems have a master, which receives the job submission from a job client
  - The master then decides how to split up the job and give it out to the workers
  - The workers do the computation on each piece and when they are done, the workers notify the master and submit the results back to them
  - The master will re-combine all of the pieces to form an output and then submit it to the job client

Approach – System Description

- Proposed simulation system is a de-centralized MapReduce system which runs on a Peer-to-Peer network
- The simulation system was implemented using Java
- Goal is a “proof of concept” to see if a MapReduce system can be implemented over a P2P network without a central command/control computer

Approach – System Description

- It will also show novel ways to recover from inefficient/disabled nodes and show techniques for handling other factors that will occur because of the P2P network
- If the system can recover efficiently from failures, this will not only eliminate any negatives of running this system on a Peer-to-Peer network, but it will yield great positives over running it on a small closed network where failures are fewer but much more catastrophic

Approach – System Description

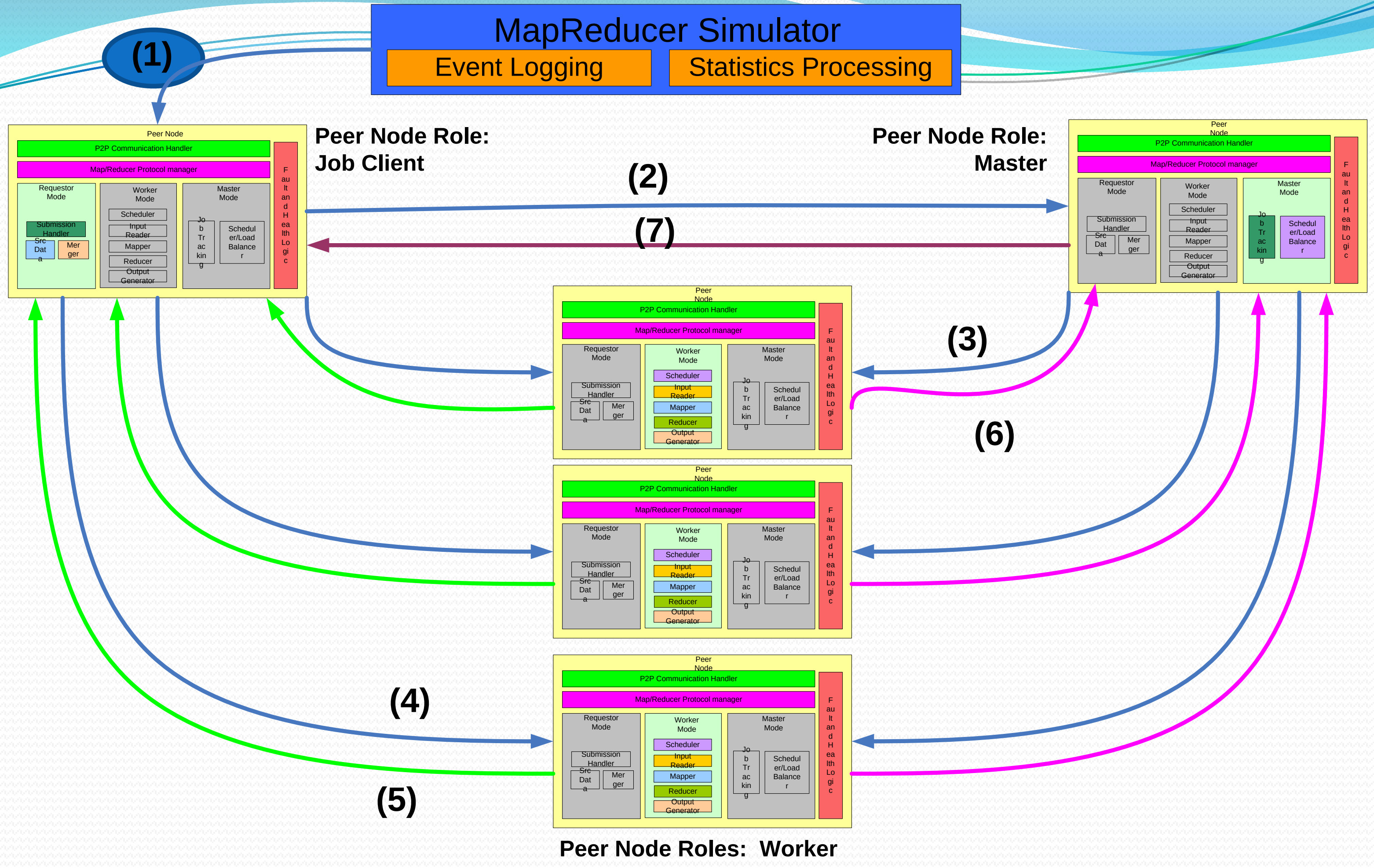
- System will consist of nodes which can take on the following roles:
  - Job Client
  - Master
  - Worker
- Each one will run through the typical job scenario, as described on upcoming slides

Approach – System Description

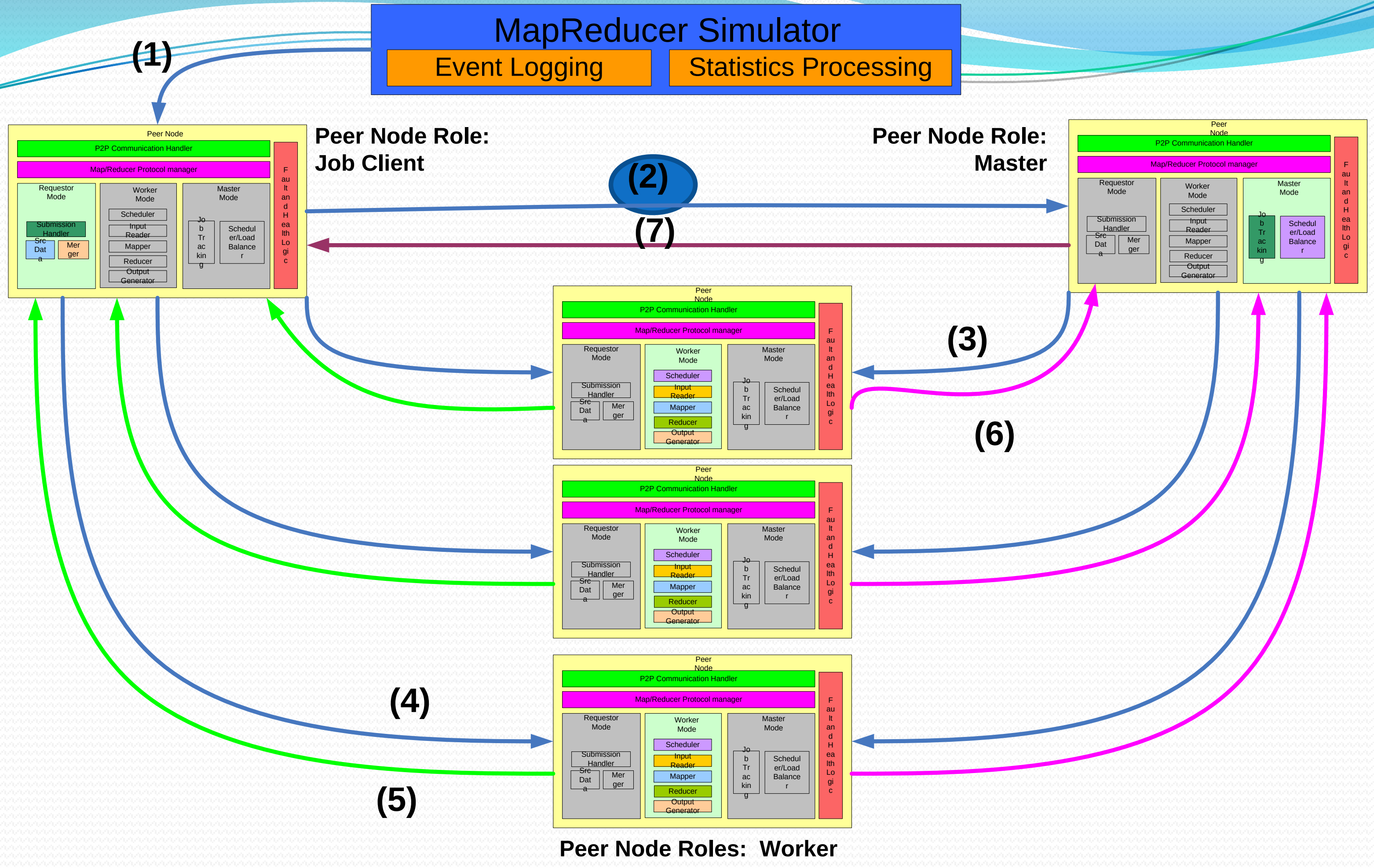
- Simulation system will be "controlled" by a simulator object which will do the following:
  - Log events and statistics
  - Control events in the simulation, such as ordering "job clients" to submit jobs and invoking the failure of different objects

Approach – System Architecture

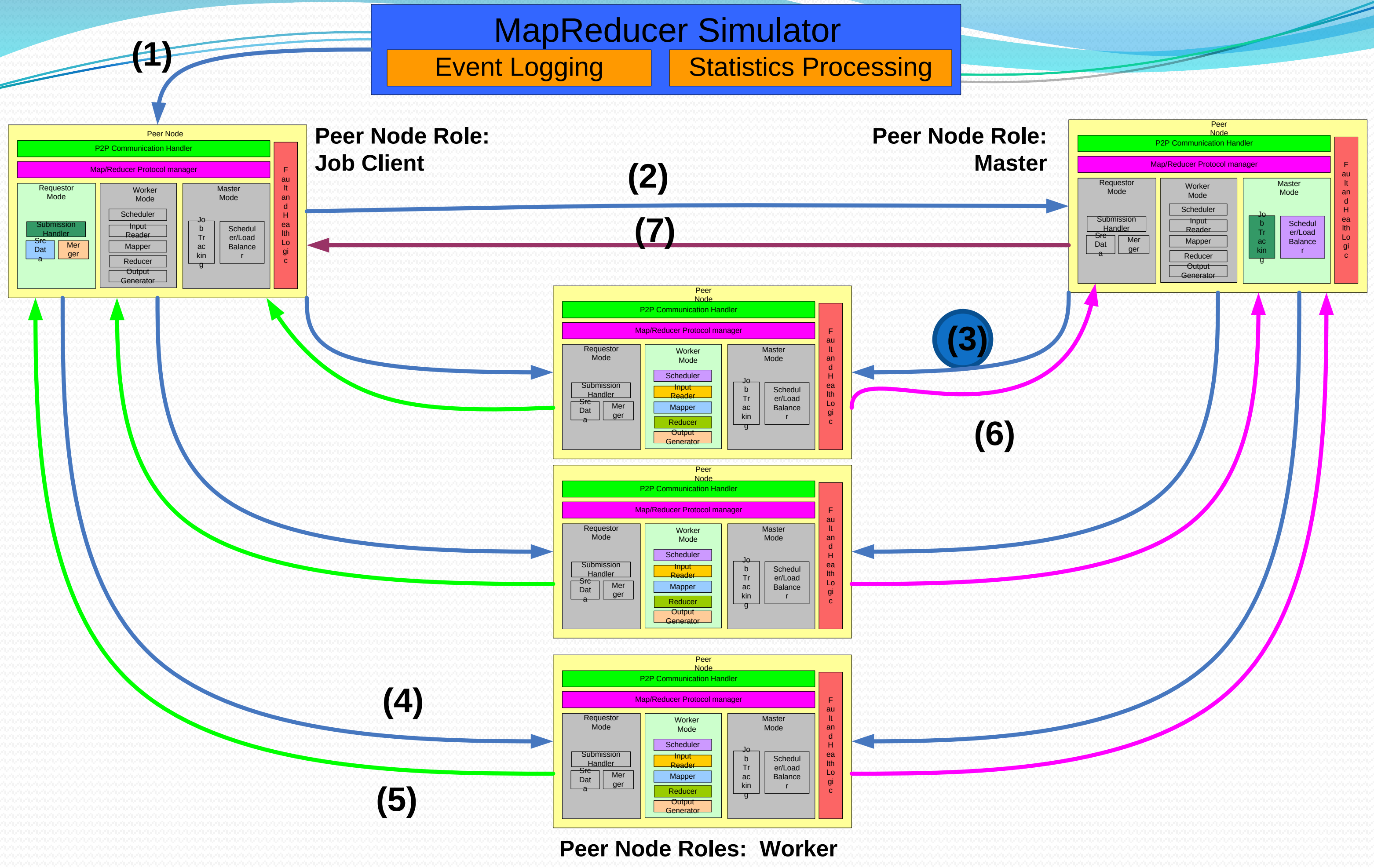
- Peer-to-Peer MapReduce simulator will be designed to simulate real conditions of MapReduce operations performed on a peer-to-peer network, but focusing more on the modified process and on the self-* optimizations to improve its performance
- In the figure on the following slides, there is a diagram which shows a rundown of one job in the de-centralized system



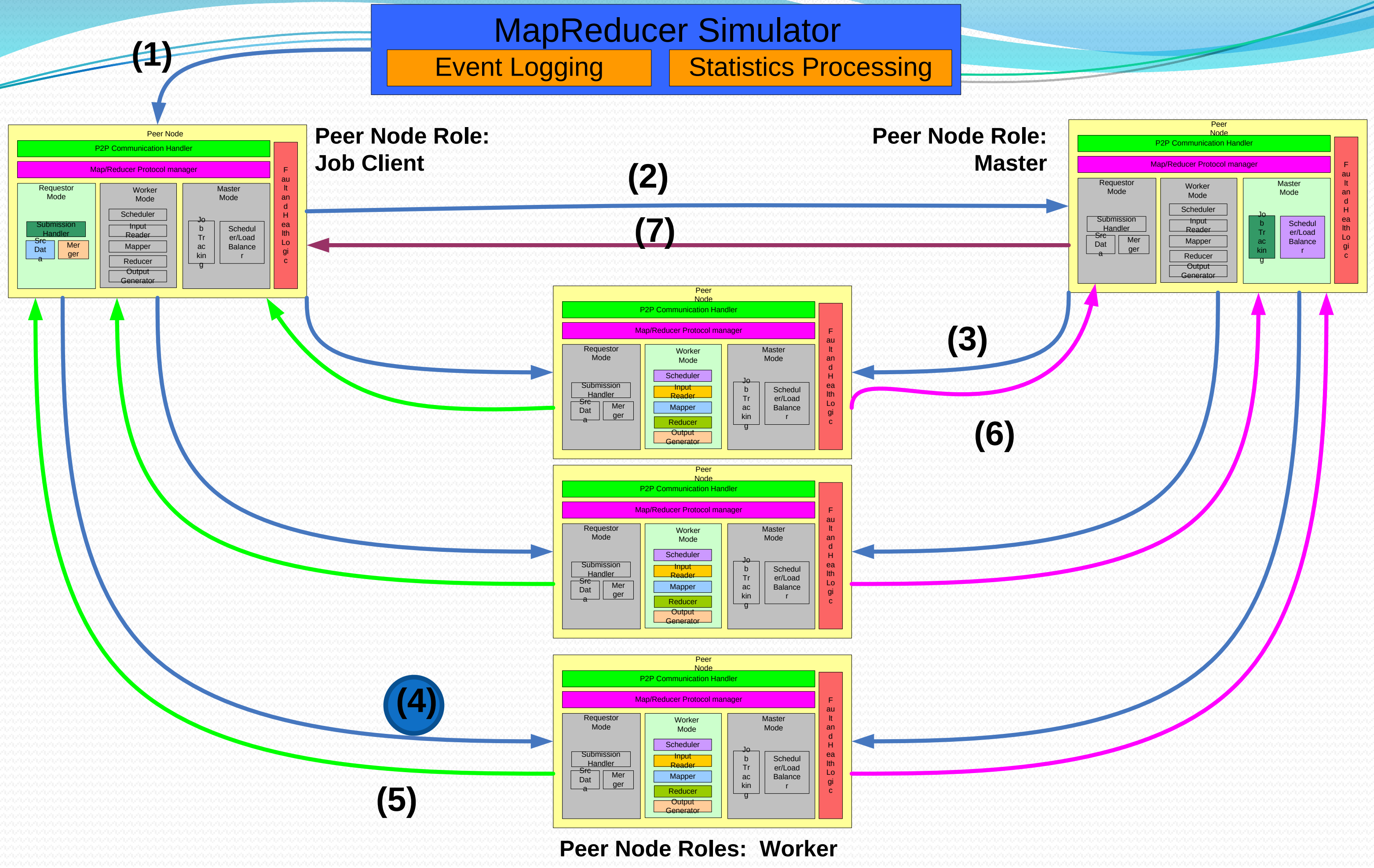
- The simulation process makes a direct call into a job client node to start a MapReduce job on a certain file.



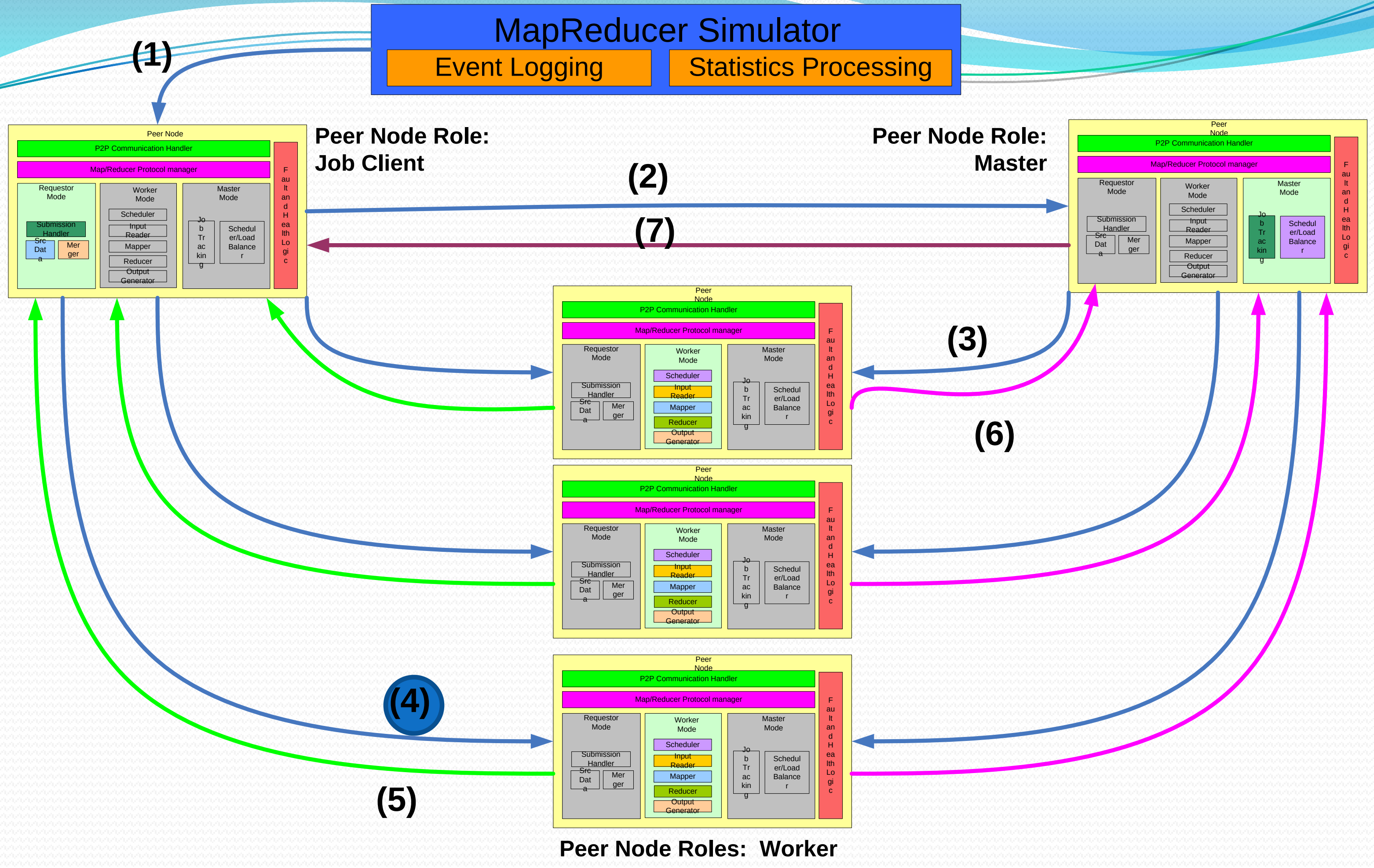
- The job client sends a message to the current master node to tell it that it wants to submit the job.



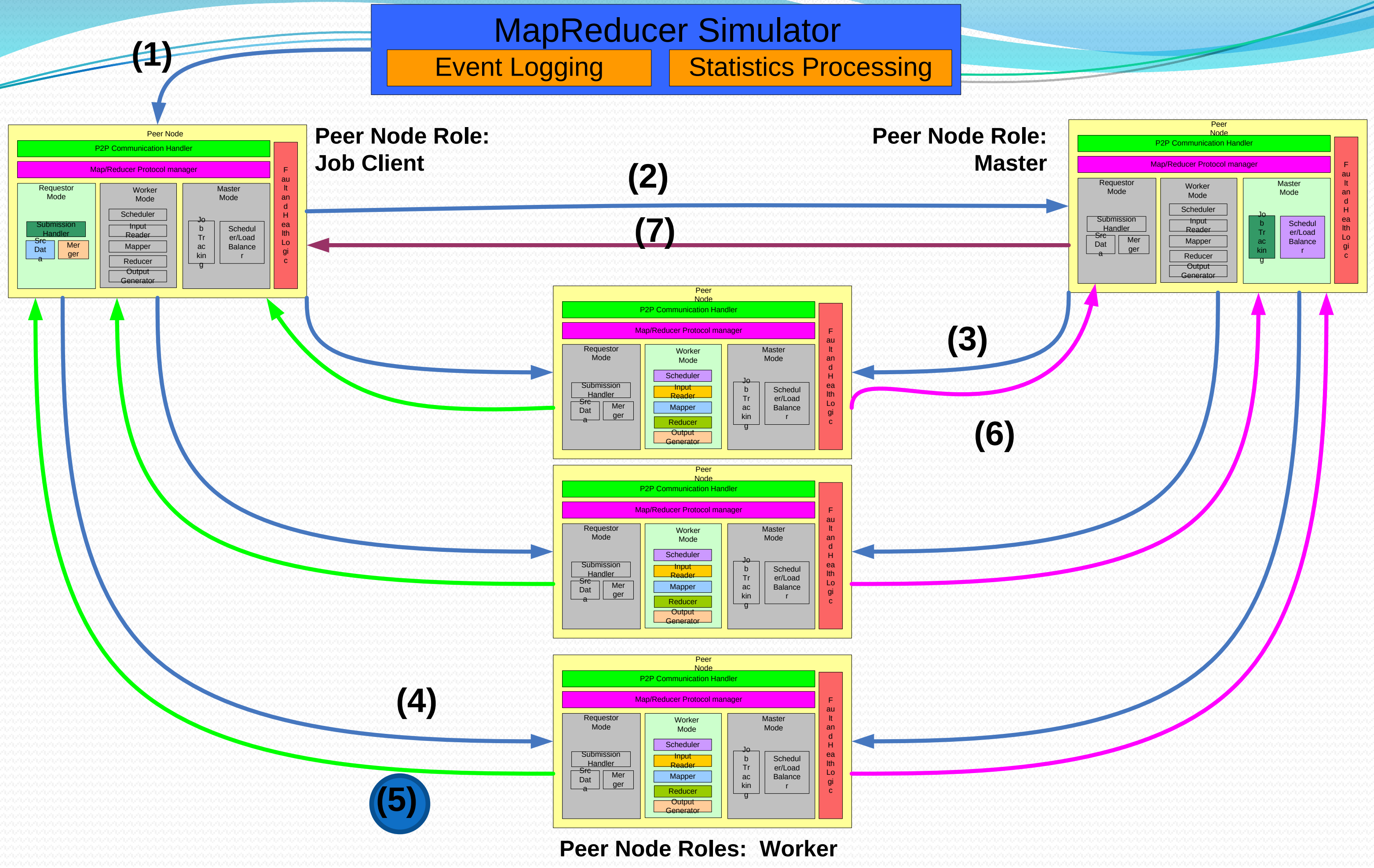
- The master analyzes the job, selects a number of worker nodes that should work on different chunks, and sends messages to each of these workers to tell them about their upcoming task.



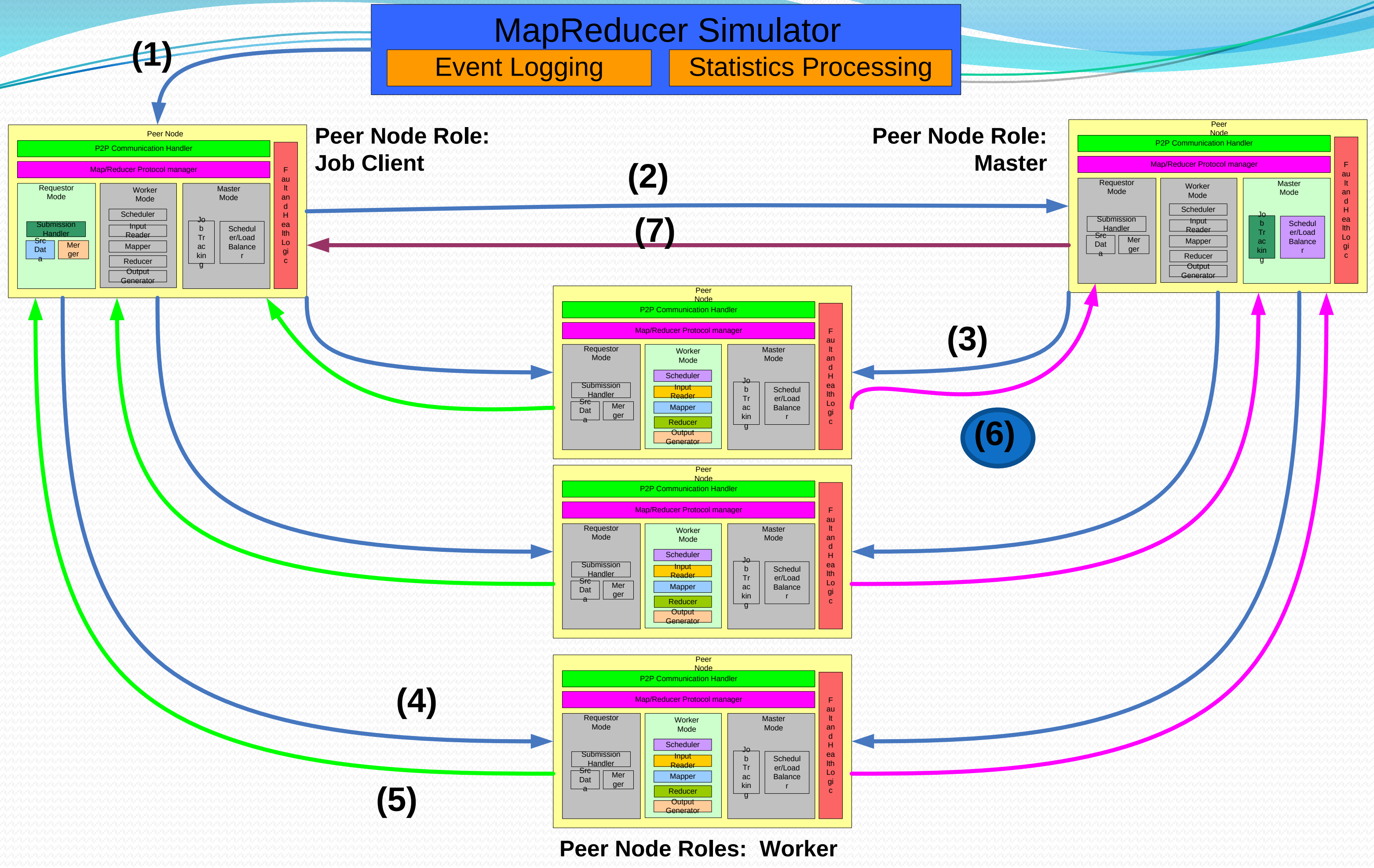
- Each worker receives the news about the chunks they are receiving. Each sends a message to the job client, which will be holding the input file the entire time, and request their chunk.



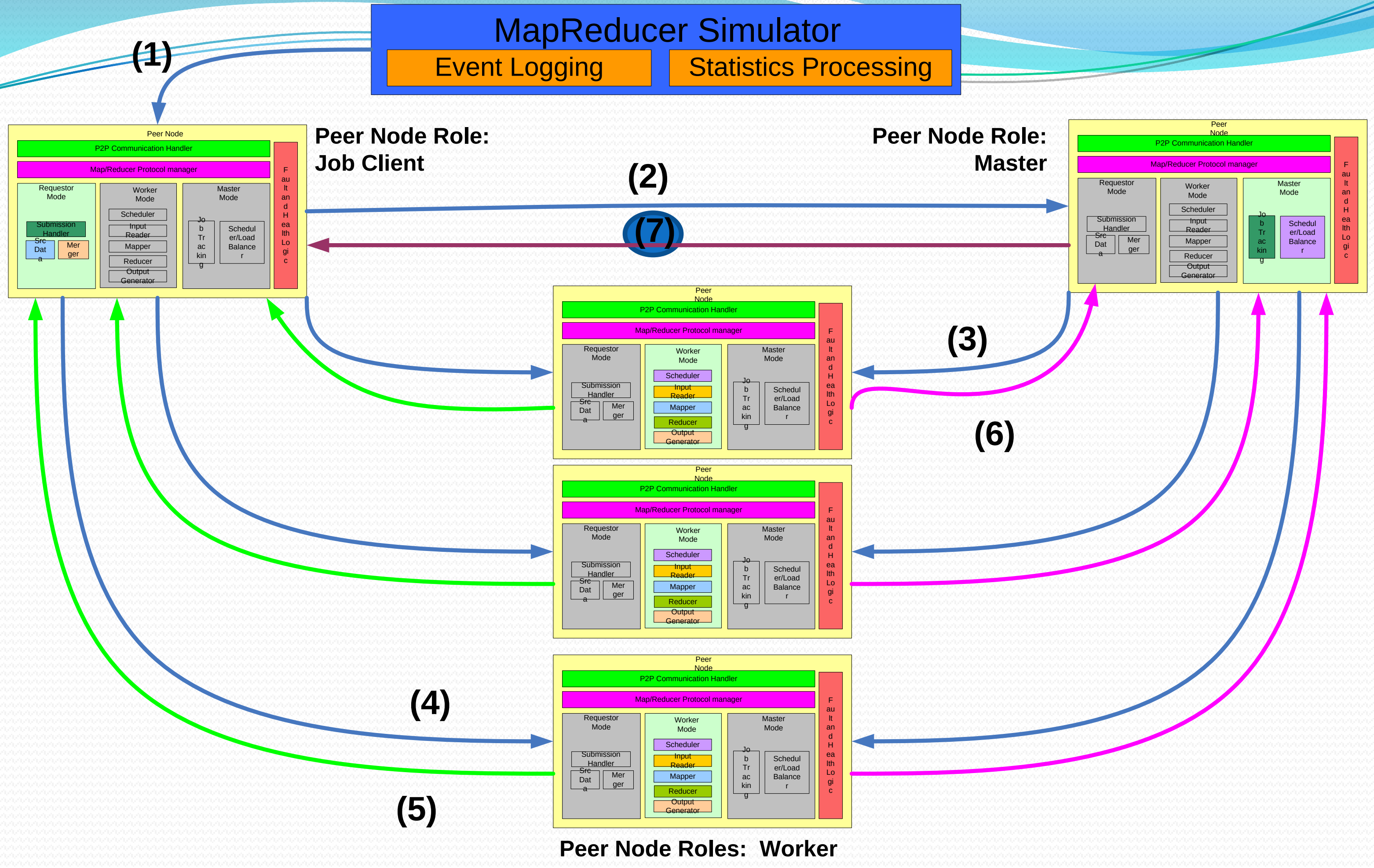
- The worker, upon receiving their chunk, will begin processing it based on how it knows to complete this type of job.



- When the MapReduce job has been completed, the worker will send the manipulated chunk back to the job client.

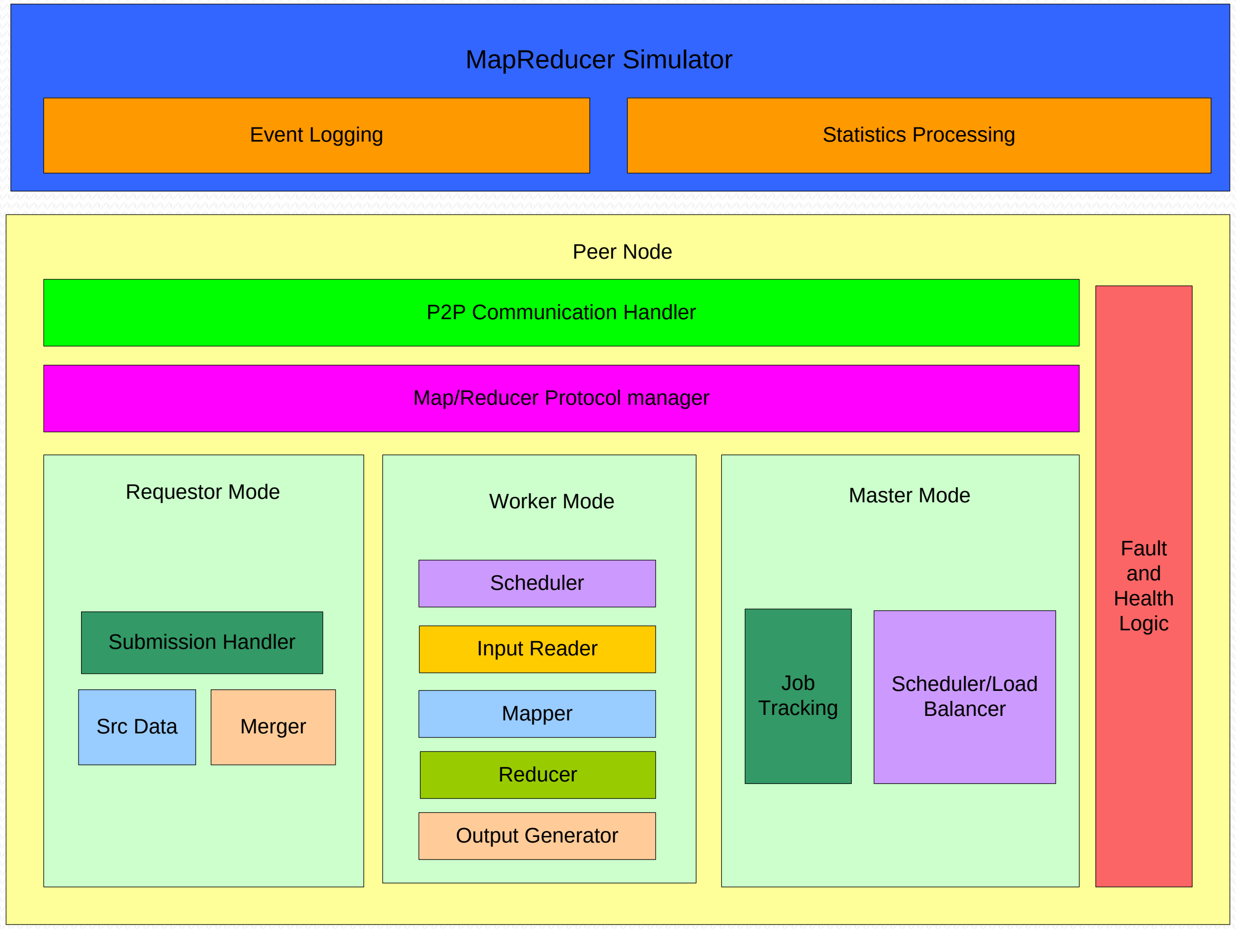


- Each worker will then send a message to the master to inform them that they have completed their MapReduce task and sent the chunk back to the job client.



- The master will send a message to the job client indicating that the MapReduce job is complete and that all chunks have been delivered back to it. The job client is responsible for merging the results of the job.

Approach – System Architecture



Approach – Self-* Algorithms

- Self-healing is accomplished by monitoring the worker nodes and the master node.
  - The master will monitor the workers by sending out a heartbeat type mechanism to all of them
  - The worker nodes will all monitor and respond to the heartbeat
  - A worker node will be declared failed by the master if it fails to respond to the heartbeat after several heart beat cycles
  - The master will be declared failed if a few neighboring worker nodes determine that they have not received a heartbeat from the master after a period of time

Approach – Self-* Algorithms

- Self-Healing (continued):
  - If a worker node fails due to loss of connectivity to the network, other fatal conditions, or from running very poorly, the failed node's computation will be re-distributed to a healthy node
  - If the master node fails, one of the other peers will take over as the master node and then continue the MapReduce operation from where it left off.
  - Therefore, the overall computation can seamlessly complete despite the failures

Approach – Self-* Algorithms

- Self-configuration is accomplished by the peer nodes negotiating who will act as the master and the remaining nodes will be dynamically allocated as workers depending on the size of the MapReduce operation
  - Workers will be recruited and selected by the master based on if they are functioning and if they are needed for the job's size

Approach – Self-* Algorithms

- Self-configuration (continued):
  - The master will have a list of workers based on an algorithm using neighbors and using broadcasts
  - In order to evaluate the effects of self-configuration, the tasks are monitored to ensure they are completed correctly even in the midst of failures, inefficiencies, and re-configurations

Approach – Future Work

- In the simulation system, each node runs as a thread under the control of the simulator and communicates via method calls, rather than sockets. In the future, these assumptions can be eliminated
- The system only focused on detecting nodes that fail or disconnect. In the future, efficiency values for each node will be computed and these will be used. Therefore, nodes can be considered "failed" even if they are connected, if they have a low efficiency value

Evaluation

- Several scenarios were developed and used to test the system
- There are scenarios to test the traditional MapReduce system against the system described here and scenarios that were run to tests the two Self-* applications and their implementation

Evaluation – Testing Basic M/R Implementation

- Tested the system without any Self-* implementation. The goal of these tests was to obtain a baseline performance in order to evaluate the effects of adding in the Self-* implementation.

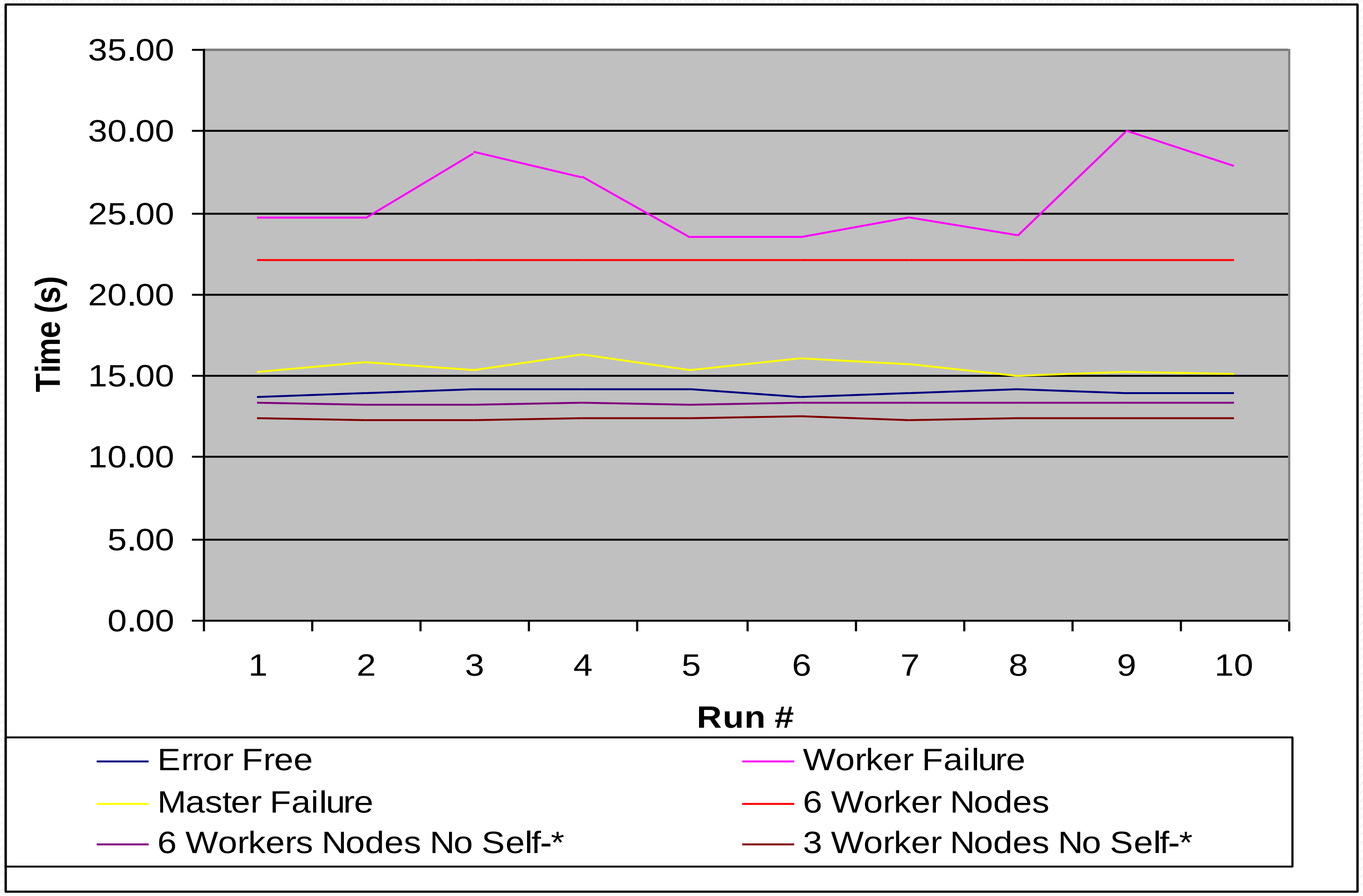
Simulation Type	**Time From Submission to Completion**
3 worker node system	12.410 Seconds (Avg)
6 worker node system	13.312 Seconds (Avg)

Evaluation – Testing the Addition of Self-*

- The second set of tests simply test the benefits of the two self-* algorithms, by running a successful scenario and scenarios with different nodes failing and self-* algorithms correcting the problems.

Scenario	**Time From Submission to Completion**
No Failures (3 workers)	13.975 Seconds (Avg)
No Failures (6 workers)	22.106 Seconds (Avg)
One Worker Failure (3 workers)	25.870 Seconds (Avg)
One Master Failure (3 workers)	15.628 Seconds (Avg)

Evaluation – Testing the Addition of Self-*



Evaluation – System Demo

- Scenario with no failures
- Scenario with Master failure
- Scenario with Worker failure

Related Work

- Hadoop - Java framework using MapReduce
  - Currently used in Yahoo web searches
  - Based on a network of computers connected via a local network rather than P2P. It also is a centralized system
- SETI@home - incorporates a centralized master node which distributes chunks of work to a P2P network of workers
  - Worker nodes can only be workers, so there is no option for recovery if the master fails
  - Jobs can only be submitted from the central system

Related Work

- Skynet - de-centralized, open source Ruby implementation of Google's MapReduce framework
  - It is adaptive, fault tolerant, and has only worker nodes
  - If the master fails, it has mechanisms to reassign its responsibilities, but only certain other nodes can take over for it
  - As a non-P2P system, tasks that fail on a particular node can only be reassigned within a defined set of workers
  - There are no outside job submissions or processing available

Related Work

- In [4], a similar topology to this paper's is described
  - The authors discuss master failure and recovery as well as worker failure and recovery
  - One key difference is that this system will only have one master. This approach will allow more available workers and still maintain the same level of functionality
  - Some of the fault tolerance ideas from [4] will be used

Conclusions

- MapReduce systems have many pitfalls, such as a centralized master, limited resources, and inability to easily recover from failures
- The de-centralized Peer-to-Peer MapReduce simulation system with self-configuration and self-healing produced by this paper’s team take large leaps towards addressing all of these issues

Conclusions

- All of the problems of a typical MapReduce system are solved by this new architecture. There are no single points of failure, the data and control (master) are de-centralized, there are limitless resources to choose from to run job submissions with the ideal number of nodes, and failures are quickly addressed
- The simulation system created and the tests and their results show that there are significant time gains from the self-* algorithms in place, under different failures. It also shows that this new system performs better with limitless nodes to choose from

Conclusions

- The work can be further extended by improving upon the team’s simulation system, implementing more components of it to remove assumptions it currently makes
- Other than improving the simulation, actual real-world Peer-to-Peer network systems and more testing and evaluation could be used to validate the results of this paper.

References

- Cardona, K., Secretan, J., Georgiopoulos, M., and Anagnostopoulos, G. 2007. A Grid based system for data mining using MapReduce. http://cygnus.fit.edu/amalthea/pubs/Cardona_Secretan_TR-2007-02_AMALTHEA.pdf
- Dean, J., and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM, Vol. 51, No. 1, January 2008,* pp. 107-113. http://www.scribd.com/doc/240523/MapReduce-Simplified-Data-Processing-on-Large-Clusters
- Hadoop, http://hadoop.apache.org/core/
- Marozzo, F.,Talia, D., and Trunfio, P. Adapting MapReduce for Dynamic Environments Using a Peer-to-Peer Model. Extended Abstract. http://www.cca08.org/papers/Poster7-Domenico-Talia.pdf
- SETI@Home, http://setiathome.berkeley.edu
- Skynet, http://rubyforge.org/projects/skynet
- Yang, H., Dasdan, A., Hsiao, R., and Parker, D. Map-Reduce-Merge: Simplified Relational Data Processing on Large-Scale Clusters. *SIGMOD'07, June 12-14, 2007, pp. 1029-1040.*