De-centralized Peer-to-Peer MapReduce System
CS647 Distributed Software Systems Project Proposal (Spring 2009)

Omar Badran
Drexel University
3141 Chestnut Street
Philadelphia, PA 19104
Ob37@drexel.edu

Jordan Osecki
Drexel University
3141 Chestnut Street
Philadelphia, PA 19104
Jmo34@drexel.edu

William Shaya
Drexel University
3141 Chestnut Street
Philadelphia, PA 19104
Wss24@drexel.edu

1. PROBLEM STATEMENT

MapReduce distributed systems are comprised of a group of servers housed at a single location. MapReduce jobs are limited to running on those servers. Depending on the size of the job, the number of available servers may not be enough to complete the job in a reasonable amount of time. MapReduce systems typically have one master and several slave or worker nodes that perform the map and reduce functionality. The master coordinates the MapReduce operation. The pitfalls of this topology are that all jobs that are run are decided within that server room, jobs can only utilize only those resources available, and that the whole system is dependent on the single master. If that master fails, the operation will not complete correctly or not complete at all.

This paper proposes the development of a simulated MapReduce system over a de-centralized Peer-to-Peer (P2P) network. This has 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; therefore, in the event that the current master fails, another peer node can take over. This will make for a more open, efficient, utilized, robust, and reliable system overall.

2. APPROACH

The proposed simulation system will incorporate self adaptation through self healing and self configuration.

Self healing will be 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 several worker nodes (exact number not yet decided) determine that they have not received a heartbeat from the master after some period of time. 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 restart or possibly even continue the MapReduce operation (where it left off). Therefore, the overall computation can seamlessly complete despite the failures. The application framework will include a module to induce random failures throughout the simulated network in order to exercise self healing.

Self configuration will be 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 by the master based on if they are functioning and if they are needed for the job's size. For example, if the job size is 100MB, perhaps two workers will be allocated, while a 1GB job may utilize 100 workers. In order to evaluate the effects of self configuration, the tasks will be monitored to ensure they are completed correctly even in the midst of failures, inefficiencies, and re-configurations. The algorithm for choosing the master node can involve using efficiency values of the worker nodes. As described more in the "Project Plan", the initial second phase will use a simple algorithm where each node will choose a random number and the node closest to zero will become master. The project's future work will include determining efficiencies of each node, allowing such a factor to be involved in this algorithm in that version.

The goal of this project will be a proof of concept to see if a MapReduce system can be implemented over a P2P network without a central command/control computer. It will also show novel ways to recover from inefficient/disabled nodes and show techniques for handling other factors that will occur only because of the P2P network. More details of the actual system design will be provided in the "Project Plan" section.

3. RELATED WORK

There are several notable MapReduce systems that exist, such as Skynet, Hadoop, and SETI@home.

Skynet is a 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.

Hadoop is a Java framework used to implement MapReduce and is currently used in Yahoo web searches. These systems are based on a network of computers connected via a local network rather than a P2P network. It also is a centralized system.

SETI@home incorporates a centralized master node which distributes chunks of work to a P2P network of workers. As a result, worker nodes can only be workers, so there is no option for recovery if the master fails. Furthermore, jobs can only be submitted from the central system.

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 between this paper's proposed system and what is described in [4] is that this system will only have one master. This approach will allow more available workers and still maintain the same level of functionality.

The system that this paper proposes has advantages over SETI@home and Skynet. The centralized topology of SETI@home leads to an inoperable system should the master fail. This paper's solution of a de-centralized system will maximize usage and availability such that any node may submit jobs, be a worker, and be the master. By incorporating the P2P architecture in the design, the paper's system is superior to the Skynet system, since the workers are not limited to a pre-defined set.

The de-centralized, P2P aspect will enhance the robustness and scalability of the system being designed. There will be more workers (and potential masters) available and jobs can be submitted by anyone. This will out-weigh any drawbacks for choosing a P2P network, such as non-locality of data with respect to the workers and the churn of peers/workers which may disappear at any time. The data flow will be discussed more in the "Project Plan" section. The issue of workers disappearing or disconnecting is the same if the system was not P2P, but just will occur much more frequently. Therefore, this will need to be a factor the system takes into account.

4. EVALUATION APPROACH

The simulator will employ statistics and event logging. The logged information will depict the state of the system when faults are introduced. For example, it will show when the master has failed and which peer node has taken over the role of master. The statistics logging will show the duration of MapReduce operations in a clean environment as well as an unstable environment. The statistics will show how the duration of the MapReduce operations are affected by various conditions.

Several scenarios will be developed and they will be used to test the system. The scenarios will be run before and after Self-* applications have been implemented. See "Project Plan" for more details. For example, here is one possible scenario:

1. Begin with 10 worker nodes and a functioning master.
2. Have a node submit a job.
3. When the job is approximately 40% complete, have a worker node disconnect. Watch that the load is balanced out among the rest of the nodes.
4. When the job is approximately 70% complete, have the master node crash. Watch for the nodes to appoint a new master and recover.

The success of the system will be judged by how it can perform different scenarios like this, all of which will rigorously test the features of the system. It will pre-dominantly test the features which make this system novel and significant as compared to the systems described in the "Related Work" section.

5. PROJECT PLAN

The paper proposes to implement this java application simulation solution in two phases.

The first phase will implement the P2P MapReduce simulation system with a single master. Upon job submission by any node in the network, work will be distributed among available peers for task completion. Self adaptation techniques will not be incorporated in the first phase. This will serve as a baseline for the concept and to ensure the basic system is operational.

Upon completion of the first phase, the team will incorporate the self adaptation discussed in the "Approach" section. Attached in the "Appendix" is the system component block diagram. Work can be broken down into "Requestor Mode", "Worker Mode", and "Master Mode." Since each of these packages is isolated, they can be split among team members. In addition, Events/Statistics, P2P Communication's Handler, and the MapReduce manager can be split as well. In phase two, fault and health logic will be implemented and the system will be tested and evaluated, using the scenarios described in the "Evaluation Approach" section.

The system created will perform the simple task of counting the number of words in a file. One node will submit the task, being a file that needs to have its words counted. The master will receive the submission and start the job by assigning an appropriate number of worker nodes and informing the worker nodes where to retrieve their data set. The workers will get the necessary input data they need from the node who submitted the job, but the entire original file is kept in that submitting node's memory space. With the submitting client storing the data file as opposed to putting the data file in a central repository, it removes the chance that all jobs would be lost with one single failure. Repository replication could be implemented but this would make the system more complex. If the submitting client fails, only data associated with that client is lost and data files from other submitting clients remain unaffected.

During processing, if any worker nodes go down, the master will re-distribute their work to the other nodes. A priority scheduler may be implemented to ensure that re-distributed work gets completed before any other jobs that the worker may be assigned to perform next. If the master goes down, the nodes will decide who the new master is and that node will take over the processing where it was left off. Due to the design of the workers retrieving their data sets directly from the submitting client, any already scheduled MapReduce operations could be completed while a new master is negotiated. When the processing is finished, the chunks will be sent to the node who submitted where they are re-combined and the node has the solution for its job. A worker node will perform the mapping and reducing of its data set while the submitting node will merge all of the "reduce" results [7].

In the initial Phase two attempt, the system will focus on only detecting nodes that fail or disconnect. This will be done using a heartbeat pinging mechanism to detect a failed worker and the absence of a ping and collaboration by the workers to detect a failed master. The "FaultAndHealth" module will be responsible for the heart beat mechanism. In a future design, overall node efficiency will be calculated. This is valuable because node work can be distributed differently based on nodes that have not failed or disconnected, but are simply performing poorly. Furthermore, when the master fails, the efficiency of nodes can be taken into account when appointing a new master. In the initial Phase two, the new master decision will be purely random.

Another reduction for the scope of this project is that in the simulation system, each node will run as a thread under the control of the simulator and communicate via method calls rather than sockets. This avoids defining messages that would be sent over TCP/IP. Furthermore, the system will assume that all nodes will be connected to each other. This avoids the process of nodes forwarding message to each other. This will allow the simulation system to focus on the Self-* solutions and handling scenarios.

6. REFERENCES

[1] 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

[2] 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

[3] Hadoop, http://hadoop.apache.org/core/

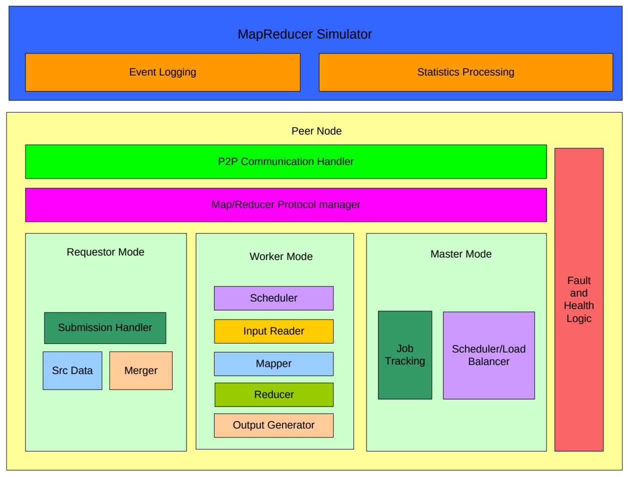
[4] Marozzo, F.,Talia, D., and Trunfio, P. Adapting MapReduce for Dynamic Environments Using a Peer-to-Peer Model. Extended Abstract. http://www.cca08.org/papers/Poster7-Domenico-Talia.pdf

[5] SETI@Home, http://setiathome.berkeley.edu

[6] Skynet, http://rubyforge.org/projects/skynet

[7] 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.*

7. APPENDIX



MapReducer Simulator
Event Logging
Statistics Processing
Peer Node
P2P Communication Handler
Map/Reducer Protocol manager
Requestor Mode
Submission Handler
Src Data
Merger
Worker Mode
Scheduler
Input Reader
Mapper
Reducer
Output Generator
Master Mode
Job Tracking
Scheduler/Load Balancer
Fault and Health Logic