De-centralized Peer-to-Peer Map/Reduce 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 and map/reduce 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 map/reduce operation. The pitfalls of this topology are that all jobs that are run are decided within that server room utilizing only those resources available and that everything 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 decentralized Peer-to-Peer (P2P) network. This has the ability to have access to numerous servers that are connected to the Internet for both extra computation ability and job submission. The decentralized 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.

2. APPROACH

The proposed 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. If a worker node fails due to loss of connectivity to the network, other fatal condition, or from running very poorly, the failed node's computation will be redistributed to a healthy node. If the master node fails, one of the other peers will take over as the master and restart or possibly continue the map/reduce 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 mappers or reducers depending on the size of the map/reduce operation. 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 goal of this project will be a proof of concept to see if a map/reduce system can be implemented over a P2P network without a central command and control computer. It will also show novel ways to recover from inefficient or disabled nodes occurring at different parts of the process. 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.

Hadoop is a Java framework used to implement MapReduce functionality and is currently used in Yahoo web searches. These systems are based on a network of computers connected via a local network versus 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 [1], a similar topology to our topology is described. The authors discuss master failure and recovery as well as worker failure and recovery. One key difference between our proposed system and what is described in [1], though, is that our 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. The paper's solution of a de-centralized system will maximize usage and availability such that any node may submit jobs, be workers, and be the Master (if there is a failure). 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 predefined set.

The de-centralized, P2P aspect will enhance the robustness and scalability of the system we are designing. 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/disconnecting is the same if the system was not P2P, but just will occur much more frequently and therefore will need to be something 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 Map/Reduce operations in a clean environment as well as an unstable environment. The statistics will show how the duration of the map/reduce operations are affected by various conditions.

Several scenarios will be developed and they will be used to test our system. For example, one possible scenario could be the following:

1. Begin with 10 worker nodes and a functioning Master.
2. Have a node submit a job.
3. Approximately 40% of the way through the job, have a worker node disconnect. Watch that the load is balanced out.
4. Approximately 70% of the way through the job, 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 all of the features of our systems. 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 solution in two phases.

The first phase will implement the P2P map/reduce 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.

Upon completion of the first phase, the team will incorporate the self adaptation discussed in the "Approach" section. Attached in Appendix 1 is our 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 the team members. In addition, Events/Statistics, P2P Communication's Handler, and Map/Reduce 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 distributing the word to each worker. The workers will get the necessary input data they need from the node who submitted the job, while the entire file is kept in that node's memory space.

During processing, if any worker nodes go down, the Master will re-distribute their work to the other nodes. A priority schedule 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. 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.

In our initial Phase 2 attempt, the system will focus on only detecting failed nodes and nodes that disconnect. This will be done using a pinging mechanism to detect a failed worker and the absence of a ping and collaboration by the workers to detect a failed Master. 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 2, the new Master decision will be purely random.

Another reduction for the scope of this project is that in the system, nodes will run as threads rather than sockets. This will avoid having to focus on messaging to handle communication. Furthermore, our system will assume that all nodes will be connected to each other. This will avoid the process of nodes forwarding message to each other. This will allow the system to focus on the Self-* solutions and not on routing issues.

6. REFERENCES

[1] Adapting MapReduce for Dynamic Environments Using a Peer-to-Peer Model

[2] A Grid based system for data mining using MapReduce

[3] Map-Reduce-Merge: Simplified Relational Data Processing on Large-Scale Clusters

7. APPENDIX