*Java-based Distributed Data Base (JDDB)*

Patrick Stickney

Department of Computer Science

University of Massachusetts Lowell

Lowell, Massachusetts

pstickne@cs.uml.edu

*Abstract* — Data is getting larger and larger as we progress into the future. With this ever increasing need for big data, we need better and more efficient ways to store, manage, and analyze this data as well.

The big data concept gained momentum in the early 2000s when organizations and industry started to collect data from every available resource [1]. By analyzing this data they can gain insight into what customers want, better ways to market to customers, boost quality and output of products, and even make more agile business decisions to promote sales.

In this paper, I will focus on the storage aspects of big data and having a low-cost distributed system is more beneficial than having a single high performance super computer.

*Keywords — Big data; Java; NoSQL; Distributed; Database*

# The Problem

## Introduction

With the flood of data being produced today, industries want to take advantage of this free resource to make data-driven decisions. Previously, these decisions could only be based on guesswork or complex models that reflected a small subset of the collected data. Data-driven decisions are important to industry because it provides them much needed insight about the target demographic. Whether we are in government, education, retail, or manufacturing big data analytics can help answer so many questions about what we can change about the future based on the past.

The question we may ask now is, how do we store such large amounts of data? For example, Walmart's customer transaction databases handle more than 1 million transations per hour holding more than 2.5 petabytes of data, eBay has close to 90 petabytes of data, Facebook handles about 600 terabytes of new data per day holding a total of about 300 petabytes of data, and Google being the data hording monster stores about 15 exabytes of data[2]! There are several answers to this question including "the cloud," RAID arrays, data warehouses, and more.

## Motivations

This is an important system to develop because it allows data exploration across a large set of data in a relatively quick amount of time. When we have a scalable storage engine with quick access times, we can then make functions that we can map to the data and apply computation and then analyze the results so that we can make decisions based on the output.

## Related Work

Some of the most notable related works to my approach have been created known as MongoDB and CouchDB. Each of these implementations have their own benefits and disadvantages for the task you are trying to accomplish.

1. MongoDB - A NoSQL database that favors JSON-like documents stored as binary files called BSON which make access times faster [3].
   * Document oriented
   * Ad Hoc Queries
   * Indexing
   * Replication
   * Load Balancing
   * Aggregation
   * Server-side JavaScript execution
2. CouchDB - A NoSQL database that uses JSON to store data in a multi-master replication architecture to scale well across multiple machines to increase performace [4].
   * Document oriented
   * Web-based access
   * Document-level ACID semantics
   * Map/Reduce Views and Indexes
   * Replication
   * REST API
   * Offline compatible

# Proposed Approach

The goal of this system is to create a document store collection that spans over multiple economic, low-cost shards. This way it provides a cost effective solution for industry to store data and process it in a distributed fashion.

My solution is broken down into three different components that come together to form a functional NoSQL database. The primary component is a node manager that is essentially the server for this application. It handles all the IO information that is exchanged between the client and database. All other components need to connect to this in order for communication to take place.

The second component is the master node. This master node is basically the client shell. It connects to the node controller and registers itself as a client. This master node has a console input for the client to run commands on which are then sent to the controller which will then run the query on each database shard in the system.

Lastly, the shard nodes are where the database collections are stored on the same or remote server or servers. Each shard has its own individual collection store that is small scale so that it does not have too much data to store or process at any given moment. I use a MapReduce approach to handling the input and output from the manager node by mapping each query to all the shard nodes and then reducing the responses from each shard node back to the client with the results from the query.

MasterNode

ManagerNode

ShardNode

ShardNode

ShardNode

# Conclusions and Future Work

For future work I would like to create an active console shell that will allow the client to program inside the shell itself to further apply computation on the resulting data. Currently, the console only sends query requests to each shard and then sends the results back to the client to visually see. With the future active-shell approach, the client will be able to not only see the resulting data but use it and store it as temporary variables saved in the console for later use.

# References

1. "Big Data." *SAS*. Web. 13 Dec. 2015. <http://www.sas.com/en\_us/insights/big-data/what-is-big-data.html>.
2. Huss, Mikael. "Data Size Estimates." *Follow the Data*. 24 June 2014. Web. 13 Dec. 2015. <https://followthedata.wordpress.com/2014/06/24/data-size-estimates/>.
3. "MongoDB 3.2." *MongoDB for GIANT Ideas*. Web. 13 Dec. 2015. <https://www.mongodb.org/>.
4. "A Database for the Web." *Apache CouchDB.* Web. 13 Dec. 2015. <http://couchdb.apache.org/>