Midterm Report

Distributed Processing System with MapReduce Model on Browsers and Node.js

101-2 Special Topics on Cloud Computing, CSIE, NTU

Tzu-Hsien Gao, Tai-Lun Tseng

Email: {r01944033, r01922094}@csie.ntu.edu.tw

0.1 Preface

This report summarizes the collective results on the project so far. It includes the original project description, survey results on other projects, roadmaps for implementation, and some possible issues for the project.

While the report is aimed to become a milestone of works done in the midterm, most parts of this report will reappear in the final project report.

Section 2 and 3 gives motivation and description of the initial idea. Section 4 summarizes the survey result of several open source projects and web-related techniques that are useful in the project. Section 5 and 6 describe the design of the system architecture, ideas of implementation and possible upcoming issues during implementation.

0.2 Motivation

MapReduce provides a distributed way for processing large-scale data. In traditional MapReduce implementations, a server node dispatches data and tasks to various client nodes and collect results after mapper and reducer functions are done by clients. Both server and client need efforts to configure connection and execution environment for MapReduce programs.

In order to get many clients in a more convenient way, we introduce a new kind of MapReduce client, i.e. browsers. In the initial scheme, researchers who need more computing resources do not necessary set up lots of clients; instead, they can use a HTTP server that establishes connection with clients visit the

researchers website (server). Data, mappers and reducers can therefore dispatch to those browsers and execute. To support a research, a resource provider only need to keep a browser opened and connected to the server. And in the future, we can extend the functionality more by enabling other node.js applications to become clients; thus researchers who are doubt for the speed of browser computing can set up client machines with node.js and libraries we provided.

0.3 Project Description

This library runs a program using MapReduce programming model on the server and dispatch mapper/reducer tasks to many registered client browsers program.

To register, clients visit a site hosted by the server and accept worker permissions. Javascripts of the site will create several HTML5 workers. They will be mappers or reducers (a browser client may has mapper and reducer workers concurrently), waiting for dispatching tasks and data from the server. Upon receiving works, the workers become active, execute the map/reduce works, return results back after finsihing execution repeatly.

Different from traditional MapReduce model, we cant communicate between browsers since the standard of communication between browsers is not popular in this time, so the the intermediate result done by map workers will be report to theserver, and then shuffle these intermediate results to the reduce workers.

Also there will be an API for node.js that provides adapters connecting server-side javascripts and node.js computing environment. Eventually the server can also dispatch tasks to node.js servers, i.e. treating node.js as a powerful MapReduce computing nodes.

0.4 Surveys on Related and Supported Works

In order to achieve our motivations, it is necessary to give a brief survey first to check how other projects implement the similar goal, and compare differences with our approach. The survey result can be categorized into two parts: *MapReduce-like implementation* and *javascript libraries and techniques*. For the first part, we try to look especially into MapReduce-like projects that is implemented in javascript.

1. Hadoop
2. Atomize.js: It is amied to provide distributed objects and atomic transaction feature between the server and all clients. The clients register to server and read/write variables with server; Atomize.js ensures that the variable is treated as "global" and identical among all clients. The descriptive sample given by authors of Atomize.js is a simple online game that uses atomic feature to manage game states. For client-server connection it uses SockJS library introduced below.
3. MapRejuice: MapRejuice is a simple implementation on MapReduce algorithm. It builds a web server that delievers javascript contents and use RESTful API as entrances for receiving outputs of map and reduce functions. It uses Socket.io for client-server connection.

4. WebWorker: WebWorker, a.k.a. Web Worker API, is the well-known feature that comes to new browsers with HTML5 standards. Specified by W3C, the objective of WebWorker is to provide concurrent programming in browsers; therefore heavy-loaded scripts can be executed in the worker thread and it does not block the main thread of browser, which leads to no-response feels to users. In our approach, it is natural to execute map and reduce functions on a worker thread in order to speed up performance in the browser.

5. WebSocket: Similar to WebWorker, it is included in the HTML5 standard. As the name shows, WebSocket provides socket programming functionalities to browsers, which traditionally rely on HTTP protocol communication. WebSocket is more suitable for modern web use cases because while HTTP protocol is stateless and disconnects after response sent/received, WebSocket can create a tunnel between the server and browsers to exchange information rapidly and simutaneously. Again it is natural for our project to use WebSocket as data exchange.

6. Socket.io: Although WebSocket provides socket-connection abilities to browsers, it currently has only plain API and unable to support old browsers. Socket.io tries to solve these problems with easy-to-use interfaces that wrap WebSocket APIs and use Ajax instead of WebSocket when older browser is detected.

7. SockJS: It is a wrapper to WebSocket. When SockJS detects that the browser is too old to provide WebSocket functionality, it uses other methods (e.g. Ajax) to emulate socket instead. This is very similar to Socket.io, which is more popular. They both have browser/server side implementation.

0.5 Architecture

Architecture

0.6 Issues

Possible issues

0.7 Reference

Reference