内蒙古科技大学

本科生毕业设计说明书

题 目：基于MapReduce的分布式计算系统

学生姓名：刘洋

学 号：1767159112

专 业：软件工程

班 级：软件2017-1班

指导教师：刘新

**基于MapReduce的分布式计算系统**

# 摘要

以 2003年，Google 发表的三篇论文为标志的大数据时代，至今已过去近二十年时间，MapReduce 那篇论文虽然只有理论，并为公开底层软件实现。但这么多年过去，Hadoop，Spark 等框架早已实现论文中所描述的功能，甚至还有所改进。

本文以 MapReduce 为基础，实现了一套基于浏览器实现的分布式系统。加之如今 Chrome 对各个平台近乎完美的兼容性，实现了一次编写，处处运行的目标。同时得力于个人移动设备的普及，手机，平板，甚至是家用游戏机，智能电视。如果急需性能，还可以通过朋友圈的方式，号召朋友们使用自己的设备，在后台开启几个标签的方式，成为计算节点，加快整体计算速度。

在 BMR 系统下，用户甚至不需要学习 C++，JAVA 等传统分布式计算用到的语言；只需要会简单的 JS，即可完成分布式计算任务的开发，开发成本极低。本文对 BMR系统的设计，以及实现时做的取舍做了详细说明，对分布式计算平台的研究具有一定的指导意义。

**关键词：MapReduce、分布式计算、高性能**

**MapReduce-based Distributed Computing System**

# Abstract

Intelligent In 2003, Google published three papers as a sign of the era of big data, nearly two decades have passed, MapReduce that paper, although only the theory, and for the public underlying software implementation. However, after so many years, Hadoop, Spark and other frameworks have already achieved the functions described in the paper, and even improved them.

This paper implements a browser-based distributed system based on MapReduce. With the near-perfect compatibility of Chrome with all platforms today, the goal of writing once and running everywhere is achieved. And thanks to the popularity of personal mobile devices, phones, tablets, and even home consoles and smart TVs. If performance is desperately needed, it is also possible to call on friends to use their own devices by opening several tabs in the background to become computing nodes and speed up the overall computing speed.

With the BMR system, users do not even need to learn C++, JAVA, and other languages traditionally used in distributed computing; they only need to know simple JS to develop distributed computing tasks, and the development cost is extremely low. This paper provides a detailed description of the design of the BMR system and the trade-offs made in its implementation, which is a guideline for the study of distributed computing platforms.

**Keywords: MapReduce, distributed computing, high performance**

# 目 录

[摘要 I](#_Toc1586389308)

[Abstract II](#_Toc1435622051)

[目 录 III](#_Toc1521037112)

[第一章 引言 1](#_Toc425407496)

[1.1 研究背景 1](#_Toc850724409)

[1.2 研究现状 1](#_Toc179020337)

[1.3 论文目标及内容 2](#_Toc170214512)

[第二章 MapRedues相关技术介绍 4](#_Toc347085380)

[2.1 分布式系统 4](#_Toc898396408)

[2.1.1 什么是分布式系统 4](#_Toc390907199)

[2.1.2 并行于分布式计算 4](#_Toc824817420)

[2.1.3 分布式系统的架构 5](#_Toc699436555)

[2.1.4 分布式系统的历史 6](#_Toc104696207)

[2.1.5 分布式系统的应用 6](#_Toc840044156)

[2.1.6 分布式系统的特点 7](#_Toc1064634514)

[2.2 函数式编程语言 7](#_Toc478529994)

[2.2.1 函数式编程 7](#_Toc327351143)

[2.2.2 lambda演算 8](#_Toc2085040434)

[2.2.3 函数式编程的优缺点 9](#_Toc636422492)

[2.3 MapReduce 编程模型 11](#_Toc1884260984)

[2.3.1 介绍 11](#_Toc1980499426)

[2.3.2 运行流程 12](#_Toc257324282)

[2.3.3 数据流 13](#_Toc1964626163)

[2.3.4 MapReduce的优缺点 14](#_Toc1910848916)

[第三章 基于MapReduce的计算框架设计和技术分析 16](#_Toc19790327)

[3.1 开发工具链 16](#_Toc1903544251)

[3.1.1 Visual Studio Code 16](#_Toc1804337198)

[3.1.2 NPM 16](#_Toc878707499)

[3.1.3 Chrome 16](#_Toc191895274)

[3.1.4 系统的技术栈 16](#_Toc1810915971)

[3.2 系统的技术栈 17](#_Toc1926479313)

[3.2.1 前端技术栈 18](#_Toc726867772)

[3.2.2 后端技术栈 19](#_Toc1579659868)

[3.3 系统的总体设计 20](#_Toc3073615)

[3.3.1 总体架构 20](#_Toc118639777)

[3.3.2 服务端模块划分 21](#_Toc1113907623)

[3.3.3 客户端模块划分 24](#_Toc1830468862)

[3.3.4 服务端客户端通信接口 26](#_Toc1986920359)

[3.4 服务端详细设计 30](#_Toc799762863)

[3.4.1 执行代码组建模块 30](#_Toc514291868)

[3.4.2 标记模块 31](#_Toc81746301)

[3.4.3 运行状态模块 34](#_Toc1668030474)

[3.4.4 检查点模块 35](#_Toc1336648580)

[3.4.5 检测模块 36](#_Toc226252793)

[3.5 客户端详情 37](#_Toc1584636761)

[3.5.1 JSON美化组件 37](#_Toc2045335680)

[3.5.2 Worker 38](#_Toc1186036231)

[第四章 框架的使用说明 39](#_Toc767722963)

[4.1 运行环境 39](#_Toc1038087965)

[4.1.1 服务端 39](#_Toc987279527)

[4.1.2 客户端 40](#_Toc1748353567)

[4.2 源码结构 41](#_Toc559658668)

[4.2.1 服务端 41](#_Toc204859216)

[4.2.2 客户端 42](#_Toc652557171)

[4.3 使用流程 43](#_Toc329387768)

[第五章 结论 45](#_Toc1954858457)

[5.1 论文总结工作 45](#_Toc953771346)

[5.2 未来展望 46](#_Toc1217071014)

[参考文献 47](#_Toc530794623)

[致谢 49](#_Toc418159123)

# 引言

## 研究背景

随着互联网和计算机技术在工作和生活中的渗透，人们的生产和生活方式都发生了巨大变化，各大社交网络平台己经成为了人们生活中不可缺少的一部分；个人移动设备的普及，大大丰富了人们的生活。对于企业来说，用户每天产生的巨量数据，使得对数据分析的需求也水涨船高。同时，企业也需要这些数据中蕴含的用户喜好等信息，以便更精准的推送广告或商品。

大多数的数据处理运算在概念上很容易理解。但由于输入的巨大规模数据量，因此要想在可接受的时间内完成运算，只有将这些计算分布在成千上万的计算机上。但处理计算，调度，异常等问题，需要大量的代码，也使原本简单的逻辑变得难以理解。

为了解决这个问题，Google 提出了一个新的抽象模型[6-8]，允许我们轻松地表达我们想要执行的简单操作，而不必担心并行计算、容错、数据分配、负载平衡等复杂的细节。这个抽象模型的设计灵感来自于 Haskell 和许多其他函数式编程语言中最初的 Map 和 Reduce 原语。其实数据处理中，大多数操作都是对数据中的信息进行提取和聚合，最终得到所需的结果。通过使用MapReduce模型，结合用户实现的Map和Reduce函数，可以非常容易地实现计算的大规模并行化。

## 研究现状

Google 在 2003 年发布的论文 《MapReduce: Simplified Data Processing on Large Clusters》[6]，并未公开内部的软件实现，但根据其论文中理论，很多大公司或者开源组织都实现了了自己的分布式计算系统。

但这些系统是仍然存在着诸多问题：架设集群费时费力，不但需要通用计算机（x86 计算机），而且需要在没有 GUI（图形用户操作接口） 的操作系统上进行搭建工作。其次传统大数据平台，用户大都需要使用 C++ 或者 JAVA 等静态类型编程语言编写执行代码，虽性能表现好，但门槛较高，对数量庞大的 JavaScript 开发者不够友好。本课题的设计方案是将浏览器作为客户端，利用浏览器提供的 Web Workers API，使得 JavaScript 这种单线程模型的编程语言，也可以完成计算密集型的任务[10]。同时提供可视化的操作界面与代码编辑窗口，操作更加方便快捷，提高了工作效率。

## 论文目标及内容

本课题采用 B/S 架构，基于 MapReduce 分布式计算平台，在 BMR 平台上为用户提供方便快捷的分布式计算功能；用户计算机，为集群提供高性能的 Worker 节点；并通过 VUE 和 ElementUI 等现代化的前端开发框架，为用户美观、易用的 WEB GUI（WEB 图形交互界面），以查看任务运行状态，节点工作情况等。为了实现上述系统开发目标，需重点研究如下几方面内容：

1. MapReduce 相关技术的研究

对 MapReduce 相关技术，如分布式系统，函数式编程，并行计算等技术展开分析，研究它们的原理，演变过程，优缺点，以及与 BMR 系统的关系。

1. 系统的设计与技术分析

介绍 BMR 系统开发过程中所使用的工具，阐述系统中各个模块的设计目的，对关键功能实现所涉及的算法进行说明，并对框架中接口的进行简要说明。

1. 基于 BMR 系统编写的计算任务的开发流程

讲解系统的部署，运行以及在该如何基于该系统，编写分布式任务，详细说明开发时需要遵循的约定。并通过几个例子讲解使用时，如何使用插件机制，提升计算效率。

# MapReduce相关技术介绍

## 分布式系统

### 什么是分布式系统

前用户将多台分布在不同区域，但通过网络通信的一组计算机，在逻辑上视为一台计算机。分布式系统的其他典型属性包括以下内容：

1、系统必须容忍单个计算机的故障。

2、系统的结构（包括网络结构，处理器结构）无法提前知道，而且在分布式程序的执行过程中，系统可能会随着时间发生变化 。

3、每台计算机对整个系统只有一个受限的、不完整的看法。每台计算机可能只知道输入的一部分。

### 并行于分布式计算

“并行计算”和“分布式计算”有很多相似之处。并行计算可以被看作是分布式计算的一种特殊形式，通过以下标准可以将并发系统粗略地划分为“并行式”或“分布式”。

1、在并行计算中，所有处理器内存时共享的，可以相互访问，以便交换信息。

2、在分布式计算中，每个处理器内存时私有的。信息是通过在处理器之间传递信息来交换的。

下图说明了分布式系统和并行系统之间的区别，图（a）是一个典型的分布式系统的示意图；连接各节点的每条线是一条连接线路，图（b）则更详细地描述了同一个分布式系统：每台计算机都有自己的本地存储器，信息只能通过使用可用的通信链路从一个节点传递到另一个节点来进行交换，图（c）显示了一个并行系统，所有处理器内存都是共享的。

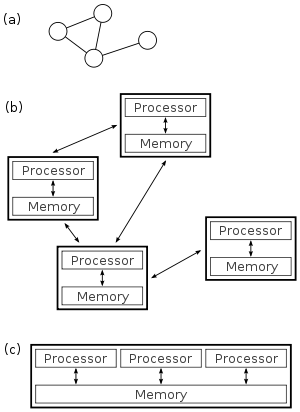


图2.1 几种分布式系统架构

### 分布式系统的架构

在较低层次上，可以使用某种技术将多个 CPU 互连起来，如 Intel 的 QPI 总线或者 AMD 的 Infinity Fabric 等技术；在更高层次上，则可以使用某种通信系统将这些 CPU 上运行的进程连接起来，如 RPC（远程过程调用） ，IPC（进程间通信） 等技术。

分布式计算架构的从另一个方面看，是在并发进程之间进行通信、协调工作的方法。通过各种消息传递协议（HTTP 或者基于 TCP 自行封装），进程之间可以直接通信，通常是主/从关系，即有明显的控制与被控制关系。或者以数据库为中心的，通过利用共享数据库，使分布式计算在没有任何形式的进程间直接通信的情况下完成。

### 分布式系统的历史

使用通过消息传递进行通信的并发进程，起源于20世纪60年代研究的操作系统架构。第一个广泛的分布式系统是局域网，如以太网，它发明于20世纪70年代。

ARPANET 是互联网的前身之一，于20世纪60年代末推出，ARPANET电子邮件发明于70年代初。电子邮件成为 ARPANET 最成功的应用，它可能是大规模分布式应用的最早的例子。除了ARPANET（及其后续的全球互联网），其他早期的世界性计算机网络包括80年代的 Usenet 和 FidoNet。

在1970年代末和1980年代初，对分布式计算的研究成为计算机科学的一个独立分支。在 2003年，随着 Google 的三篇论文（MapReduce，GFS，Bigtable）的发表，标志着分布式系统正式进入大规模商用的阶段。

### 分布式系统的应用

使用分布式系统可能包括如下几个原因：

1、数台由物理位置分隔的计算机，需要通过网络进行通信来完成某项工作。例如现在企业中流行的异地容灾，防止自然灾害等不可抗力导致的数据丢失。

2、在大多数情况下，使用单台计算机原则上是可行的，但由于实际原因，使用分布式系统收益更大。例如，与单台高端计算机相比，使用几台低端计算机组成的集群来获得所需的性能水平，成本更低。

3、分布式系统可以比非分布式系统提供更高的可靠性，因为不存在单点故障，即整个系统可以允许多个几点故障的同时，不影响服务。

4、分布式系统可能比单一的单处理器系统更容易横向扩展，一个设计良好的分布式系统，可在遇到性能瓶颈时，通过增加节点的方式达到所需性能要求。

### 分布式系统的特点

CAP定理（CAP theorem）指出，对于一个分布式计算系统[17]来说，不可能同时满足以下三点：

1、一致性（Consistency）：每一次读取都会收到最近一次的写入，否则就会出现错误。

2、可用性（Availability）：每次请求都能收到响应（非错的），但是不保证获取的数据为最新数据

3、分区容错性（Partition tolerance）：尽管节点之间的网络传输丢弃（或延迟）了任意数量的信息，系统仍能继续运行。

根据该定理，一个分布式系统最多只能满足其中两个。

而现代分布式系统中，一般选择优先保证可用性（Availability）和分区容错性（Partition tolerance），而选择牺牲一致性，只保证最终一致性，即只保证随着时间的迁移，不同的节点上的同一份数据会逐渐趋同，而不保证同一数据，在任何时刻都相同。

## 函数式编程语言

### 函数式编程

在计算机科学中，函数式编程（Functional Programming）[19]是一种编程范式，程序是通过应用（applying）和组合（composing）函数来构建的。它是一种声明式的编程范式，其中函数定义一颗由表达式构成的树，将值映射到其他值，而不是更新程序运行状态的命令式语句的序列。

在函数式编程中，函数被视为一等公民，这意味着它们可以被绑定到名称（包括局部标识符），作为参数传递，并从其他函数中返回，就像其他数据类型一样。这使得程序可以以声明性的方式编写，这也使得小的函数以模块化的方式组合成为可能。

函数式编程有时被视为纯函数式编程的同义词，纯函数式编程是函数式编程的一个子集，它将所有的函数视为确定性的数学函数，或称纯函数。当一个纯函数被调用并带有一些给定的参数时，它将总是返回相同的结果，并且不会受到任何可变状态或其他副作用的影响。这与命令式编程中常见的不纯程序形成对比，后者可能有副作用（修改程序的状态）。纯函数式编程的支持者声称，通过限制副作用，程序可以减少错误，更容易调试和测试，也更适合于形式化验证。

### lambd演算

函数式编程的基础是 lambda 演算——一个数理逻辑中的形式系统，它是一个通用的计算模型，由数学家 Alonzo Church 在 1930 年代引入，lambda 演算中只有三个基本的语法。

表2.1 lambda语法描述

|  |  |  |
| --- | --- | --- |
| 语法 | 名称 | 描述 |
|  | 变量 | 一个表达式可以是单个字母，或多个字母。一般的，我们可以把两个表达式写在一起组成一个新的表达式。 |
|  | 抽象 | 函数定义（M 是一个 lambda 项）。变量 x 绑定在表达式 M 中。 |
|  | 应用 | 将 N 作为参数，应用到函数 M 上 |

lambda 演算的能力十分强大，例如定义数字

（6-1）

（6-2）

使用这个函数计算 0

（6-3）

相似的

（6-4）

（6-5）

在此基础上，可以定义加法

（6-6）

代入 1 和 2

（6-7）

化简后可得

（6-8）

即为 3

而且 lambda 演算与图灵机是等价的，意味着我们使用传统编程语言编写的计算机程序，理论上都可以转换使用函数式编程语言重新实现。

### 函数式编程的优缺点

函数式编程是一种写出可被证明的正确的代码的方式。有如下优点：

1、对于相同的参数，其返回值是相同的，它的调用没有副作用（不会更改其他变量或是 I/O 设备的状态）。编程时不需要关注整个系统状态，也能更好的与其他模块进行解耦合。

2、测试友好。现代软件工程中，测试十分重要，同时也是保证系统稳定，正确的重要手段。函数式编程中的纯函数，对于任何给定的输入，输出总是相同的。这提升了可预测性。测试时只需要准备测试集和对应结果，进行对比，无需关注除输入之外的任何状态。

3、函数式编程导致更少的 BUG。因为每个函数都是输入和输出的简单映射，在每个层次上进行简单的堆栈跟踪或打印语句就能发现问题。在命令式编程中，在代码库的任何地方都可能存在共享的或可变的状态，这些共享或可变的状态，十分容易导致错误。

4、代码十分容易进行复用。在面向对象编程中，因为类的设计的问题，可能出现 “需要一只猴子，却引入了整个动物园” 的情况。而函数式编程中，各个函数相对独立，对代码复用更加友好。

5、并发编程时更加安全。纯函数在定义上是线程安全的。当我们能够保证没有两个并发的进程同时试图访问相同的数据时，代码就是线程安全的。这被称为 race conditions（条件竞争）[5]，在实际编程中，是最难解决的错误之一。但纯函数从不与程序的其他部分共享状态，所以它们不可能有条件竞争。

同时，因为在传统的函数式编程中，过于强调变量的不可变性，使得：

1、输入/输出(IO)

IO依赖于 side-effect（副作用），所以它本质上 non-functional 的。但平常编程中，几乎不存在没有 IO 的程序。虽然有很多方式加入 IO，如 monad，但理解难度较高。

2、递归

递归通常被称为函数式编程的最佳部分之一，但递归的使用代价非常高。他们经常利用调用栈作为数据结构，但内存是一种宝贵的资源。有一些写递归函数的方法，在内存方面没有这么高的成本，但它们要么需要编译器优化，要么以截然不同的风格重写递归函数。

3、术语

函数式编程有很多优点，但易于理解并不是其中之一。由于它植根于数学，它可能会被许多术语所困扰。像“纯函数”和“透明引用”这样的术语，对于那些刚接触函数式编程的人，甚至对于那些还没有学会所有术语的函数式老手，理解起来十分困难。

4、抽象

由于函数式编程对函数的处理方式十分灵活，在没有注释的情况，理解代码将十分困难。在使用函数式编程的项目中，对于一些关键函数，注释长度远长于代码长度的情况，十分常见。以至于一些现代函数式编程语言，如 Haskell，注释的语法甚至包括 latex。

## MapReduce 编程模型

### 介绍

MapReduce 是一种运行在集群上，处理大规模数据的编程模型。MapReduce 所采用的编程模型源自于函数式编程里的 Map 和 Reduce 原语。

Map 函数的输入和输出都是键值对(Key-Value pair）起到类似提取信息的作用。

Reduce 函数的输入是同一 Key 值的所有 Value，输出是键值对。Reduce函数的作用是对 Map 提取的信息，进行聚合。

MapReduce 是一个简约的编程模型，将一个 Job 以分割输入的方式，分成多个小的 Task。使得处理每个 Task 的 Worker 的要求大大降低。因此可部署在廉价服务器上，随着集群规模的扩大，内存、磁盘、网络等出错是常态。不过在 MapReduce 编程模型里，Job 的可分割性与 Map 函数 和 Reduce 函数的无状态性。Worker 节点的出错对集群整体的计算性能不会有太大影响，只需要将该 Task 交付给其他正常工作节点重新计算即可。

### 运行流程

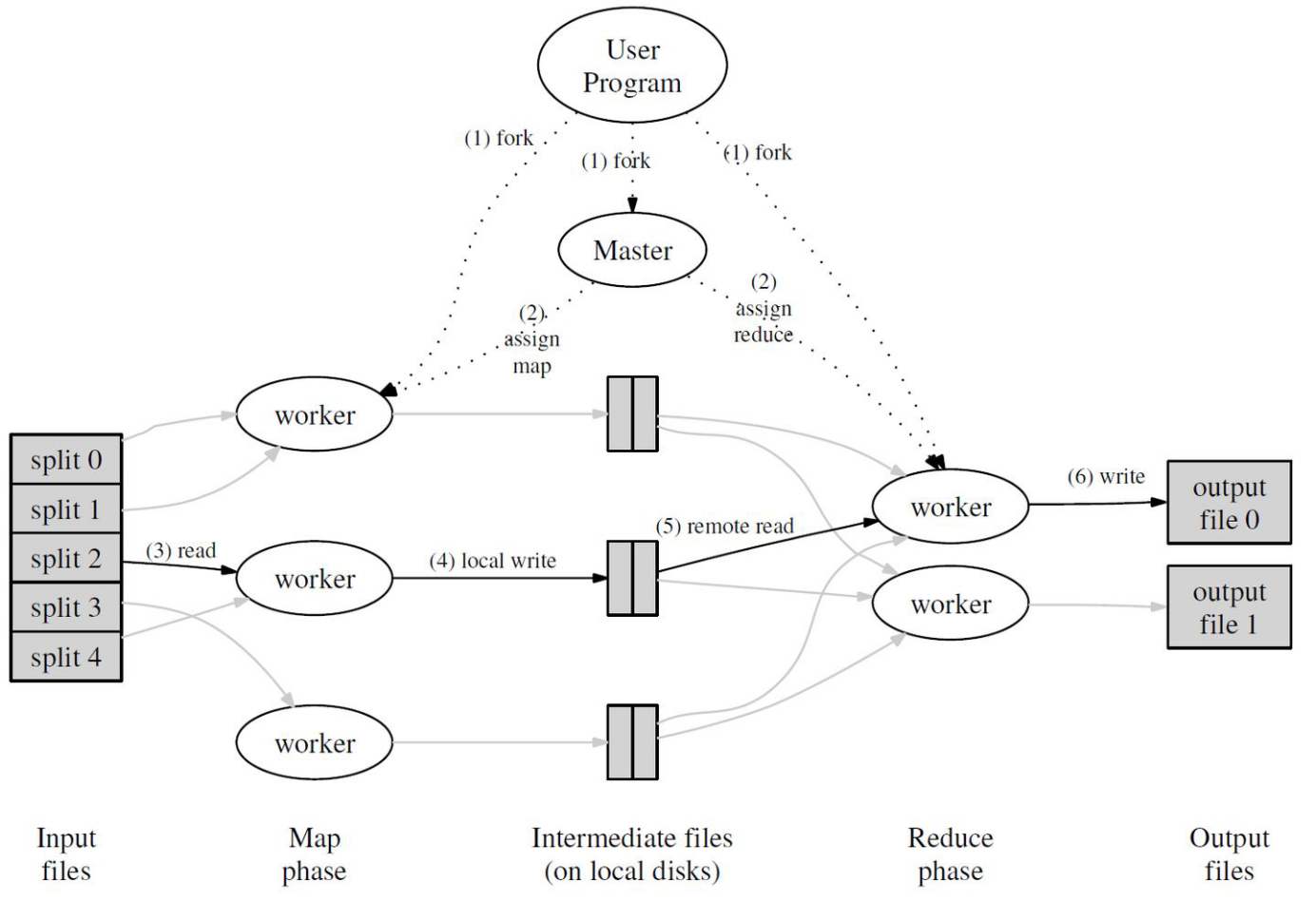


图2.2 分布式运行流程

首先，用户编写 Map 函数和 Reduce 函数，然后提供运行配置，分片数量 M，各类型 Worker 数量，如果有需要，还可变现 Partition 函数。

MapReduce 开始计算后，整个过程如下：

1. Master 选择 Map 的 Worker 们（Mapper）开始，传输分片，之后将读入的内容转换为 Map 函数的输入类型，即键值对（Key-Value pair）并调用自定义的 Map 函数。由 Map 函数产生的中间结果，即代表 Map 函数从中抽取的信息会被临时存储在 Worker 的内存中。

2. 如果用户有自定义 Partition 函数，将会在上述工作进行，Mapper 们会周期性的调用用户指定的 Partition 函数。任务完成时，Mapper 便会将中间结果（或者在硬盘中的地址）传回给 Master。

3. Mapper 计算的中间结果，在排序之后。会传给 Reduce 的 Worker 们（Reducer）会对读取到的数据进行排序以使得相同键的键值对能够连续分布

4. 之后，Reducer 会为将每一个 Key 相等的 Value 的集合传入用户自定义的 Reduce 函数。Reduce 函数执行的结果传回 Master，或写入最终的结果文件。

随着集群规模的扩大，而由廉价服务器构成的节点，随时可能出现故障，所以 Master 还会有周期性的生成 checkpoint（检查点），该检查点会记录每一个 Map 和 Reduce 任务的状态（完成，未完成等），使得即使 Master 故障后，也能快速恢复运行状态。同时 Master 会周期性的对所有的 Worker 节点进行 Ping，看其是否还在保持运行，以保证计算任务的顺利进行。

### 数据流

每个成熟的模型都应该坚持开放-封闭的原则，代码应被划分为不可变点和可扩展点。MapReduce 计算模型中的不可变点是排序部分，而热点是由应用程序定义的：

1、Input reader

将 Job 的 slice（切片），转换成 Map 函数对应的输入类型

2、Map 函数

函数的输入和输出都为键值对(Key-Value pair）起到类似提取信息的作用。

3、Partition 函数

每个 Map 函数的输出都被分配给了某个特定的 Reducer，以达到分片的目的。Partition 函数给定了 Key 和一组特定的 Reducer，并返回对应 Reducer 的索引。即更改 Map 函数的 Key，如在抓取网页的任务中，需要将同一个域名的结果，交给同一个 Reducer 处理，此时的 Partition 函数即为 hostname(Key)

4、Compare

所有的 Key 都会经过依次排序，以使得相同的 Key 的数据在文件中连续，而排序函数可以使用用户自定义的 Compare 函数比较 Key，从而达成自定义排序的效果，如比较用户自定义的 Key，如域名等。

5、Reduce

Reduce 函数的输入是同一 Key 值的所有 Value，输出为键值对。Reduce函数的作用是对 Map 提取的信息，进行聚合。

6、output writer

将 Reducer 函数的输出，转换成可写入的格式，并写入到非易失存储器中。

### MapReduce的优缺点

优点：

MapReduce 易于编程。只需要简单的实现一些函数，就可以获得分布式系统的高性能和容错性，与写一个简单的串行程序并无二支。也因为这个原因，使得 MapReduce 的使用人员十分多。

良好的扩展性。当性能无法满足要求时，只需要简单的进行横向扩展就可提升计算能力。

高容错性。得益于大量引入的函数式编程特性，使得节点的故障，对整个集群的运行无太大影响，并通过重试的方式，保证计算任务的正确性。整个过程无需人工干预。

缺点：

不擅长实时计算。将输入分片的方式，进行并行，使得该编程模型只适合离线计算，而对实时性要求强的在线运算无能为力。

## 本章小结

本章对使用技术进行了基本的介绍，并结合发展历史，分析了各个技术的优缺点。并对未来的发展进行了展望。

# 基于MapReduce的计算框架设计和技术分析

## 开发工具链

### Visual Studio Code

Visual Studio Code 是微软开发的跨平台源代码编辑器。原生功能虽然只有简单的文本编辑功能，但通过插件机制，可以支持包括支持调试、语法高亮、智能代码完成、片段、格式调整，代码静态检查等功能。

### NPM

NPM（最初是Node Package Manager的简写）是一个用于 JavaScript 编程语言的包管理工具。NPM 是 JavaScript 运行环境 Node.js 的默认软件包管理工具。它由一个命令行客户端（也叫 NPM）和 NPM 仓库组成。

### Chrome

Chrome 浏览器是一个由 Google 开发的跨平台网络浏览器。凭借其超高的市场占有率，已成为浏览器领域的事实标准。借助其内置的 V8 JavaScript Engine，让 BMR 系统使用一门动态语言开发成为了可能。

### 系统的技术栈

Chrome 浏览器是一个由 Google 开发的跨平台网络浏览器。凭借其超高的市场占有率，已成为浏览器领域的事实标准。借助其内置的 V8 JavaScript Engine[18]，让 BMR 系统使用一门动态语言开发成为了可能。

## 系统的技术栈

在计算机的领域中，技术栈或解决方案栈是一组软件子系统或组件，来创建一个完整的平台，这样就不需要额外的软件来支撑应用程序。一般说应用程序运行在所产生的平台之上。

例如，为了开发一个 WEB 应用，架构师一般将技术栈定义为目标操作系统、网络服务器、数据库和编程语言。技术栈的另一个说法是操作系统、中间件、数据库和应用程序。通常，技术栈的组件是由不同的开发者独立开发的。

一个整体系统中的一些组件/子系统经常被选择在一起，以至于这套特定的系统被称为代表整体的名字，而不是通过命名各部分。通常情况下，这个名字是代表各个组件的首字母缩写，如 SSM（Spring、SpringMVC、mybatis）这一 JAVA 开发中常用的技术栈。

BMR 系统是一个 WEB系统，所以和常规的 WEB 项目一样，分为前端和后端；前端使用的技术栈为业界常用的 VUE + ElementUI 的组合，代码编辑用的组件为微软提供的monaco-editor。而与后端通信，因为 MapReduce 的 Master 需要主动向 Worker 发送消息，所以使用了 HTTP 协议 和 WebSocket 协议。后端则使用了 Node.js[11] 的运行环境，和 Express 的开发框架[13]， ws 提供的 WebSocket 协议[14]，与前端进行通信。由于 MapReduce 涉及到的检查点，记录 Worker 状态等，传统数据库难以表示，所以使用了自定义文件，配合定制化的序列化和反序列化协议完成。

### 前端技术栈

#### Vue

全称是 Vue.js，是一个开源的模型-视图-视图模型的前端 JavaScript 框架[20]，用于构建用户界面和单页应用程序。

Vue.js的特点是一个可逐步适应的架构，专注于声明式渲染和组件组成。核心库只关注视图层。复杂应用所需的高级功能，如路由、状态管理和构建工具，通过官方维护的支持库和包提供。

Vue.js可以使用称之为指令的 HTML 属性来扩展 HTML。指令为 HTML 应用程序提供功能，可以是内置指令，也可以是用户定义的指令。

#### ElementUI

一个基于 Vue.js 的，由阿里巴巴饿了吗团队开发的开源组件库，在易用性，可维护性，可拓展性上较为成熟。且使用组件库可以帮助设计者坚持的产品的准则，它对开发的影响较小。你不是在重新创建组件，而是在重复使用组件。易于定制将帮助设计师更加灵活。

且组件库可以很好的实现设计，使用的分离，使得设计师专注于视觉风格，不用关注排版，逻辑等无关信息。而前端程序员可以专注内容和逻辑，与视觉风格解耦。不仅如此，还可以让擅长于写 CSS 的程序员，与使用 JS 实现易用性，扩展性等功能的程序员，各自实现自己擅长的部分。

#### HTTP协议

HTTP 是一个应用层协议，一般依赖于一个底层的、可靠的传输层协议，通常是 TCP。然而，HTTP也可以被使用不可靠的协议，如UDP，例如在HTTPU和简单服务发现协议（SSDP）。在客户-服务器计算模型，即 CS 模型中，HTTP作为一个请求-响应协议发挥作用。

#### WebSocket协议

WebSocket是一种计算机通信协议，通过单个TCP连接提供全双工通信通道。WebSocket协议使网络浏览器（或其他客户端应用程序）和网络服务器之间的互动比HTTP轮询等半双工替代方案的开销更低，有利于从服务器传输和到服务器的实时性强的数据。

### 后端技术栈

#### Node.js

Node.js是一个开源的、跨平台的 JavaScript 运行环境，运行在由 Google 在 Chromium 项目中的一个模块，V8 引擎之[18]上。给 JavaScript 提供了一个浏览器之外的宿主环境。同时 Node.js 提供了文件I/O、网络等核心模块来创建各种 Web 服务器。

Node.js与其他编程语言最显著的区别是非阻塞的 IO，配合 async 等特性，以极短的代码，便可编写出高性能的 WEB 服务器，同时还不失可读性。

#### Express

Express.js 是一个最小化、且灵活的Node.js网络应用程序框架，该框架提供了如下核心功能。

允许设置中间件来响应 HTTP 请求。

定义了一个路由表，用于根据 HTTP 方法和 URL 执行不同的操作。

允许在向模板传递参数的基础上动态地渲染 HTML 页面。

#### ws

ws 是一个简单易用、速度极快、经过全面测试的 WebSocket 客户端和服务器实现。并且通过了相当广泛且包含服务器、客户端的自动化测试套件。

## 系统的总体设计

BMR 系统采用模块设计，它强调将程序的功能分离成独立的、可互换的模块，这样每个模块都只包含了某个功能执行所需的东西，从而在模块之间进一步解耦。

### 总体架构

BMR 系统是一个以 MapReduce 理论为基础的，利用 WEB 浏览器作为客户端，Node.js 作为服务端运行环境，无数据库的 WEB 项目。所以整体架构与传统的 WEB 项目类似，分为运行在浏览器上，直接面向用户的前端和运行在服务器上的后端。

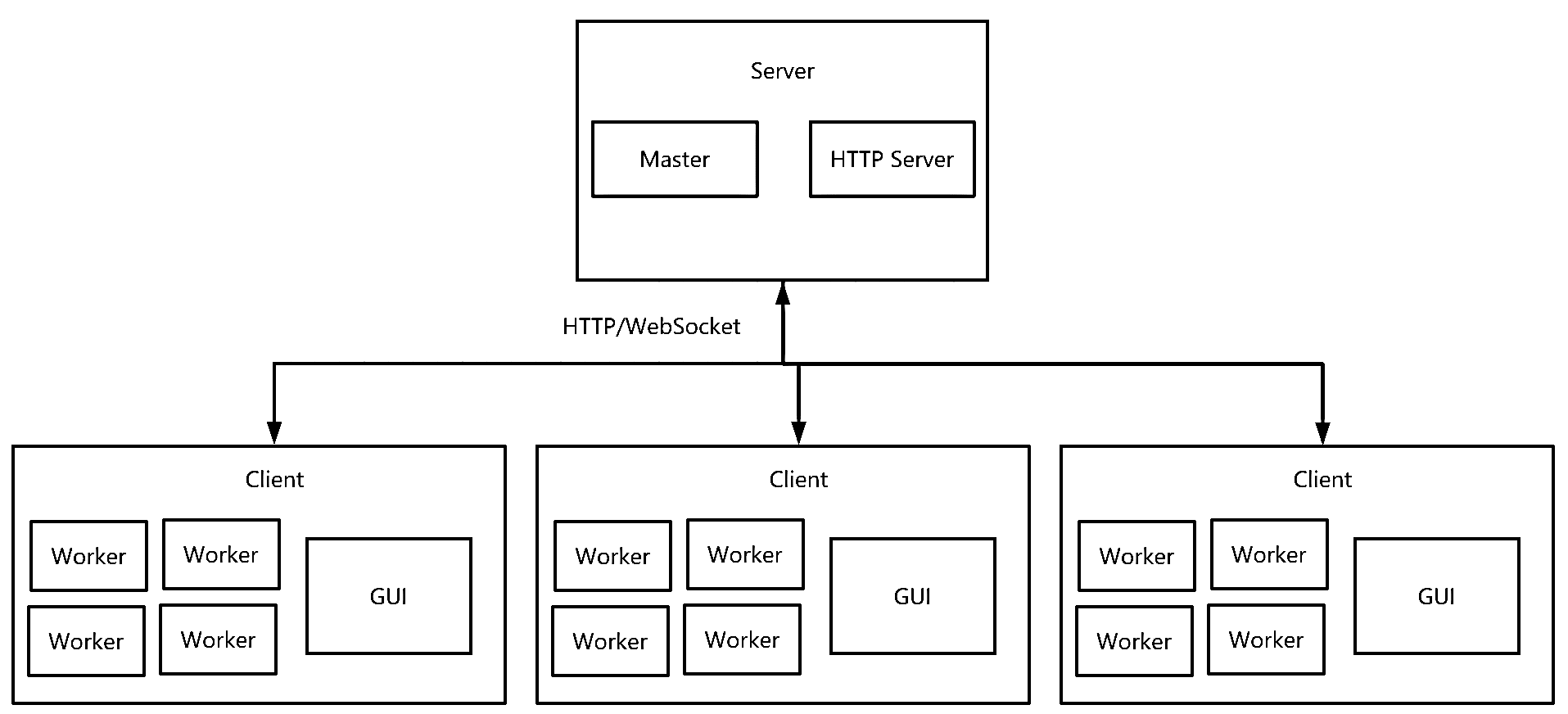


图3.1 整体架构图

### 服务端模块划分

服务端主要分为两部分，一个是传统 MapReduce 系统中，Master 的部分。另一个则是为前端提供必要信息的 HTTP Server。

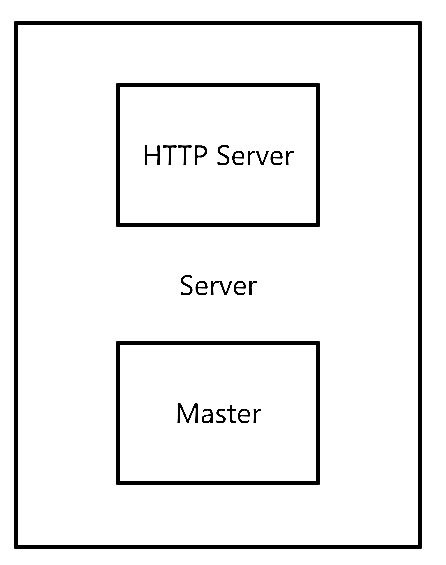


图3.2 服务端架构图

#### HTTP Server

负责 Server 的主要工作，包括提供 HTTP 接口。提供前端的静态文件，使得用户通过浏览器访问时，可以拿到可供浏览器渲染页面的文件。

主要由以下几个模块组成：

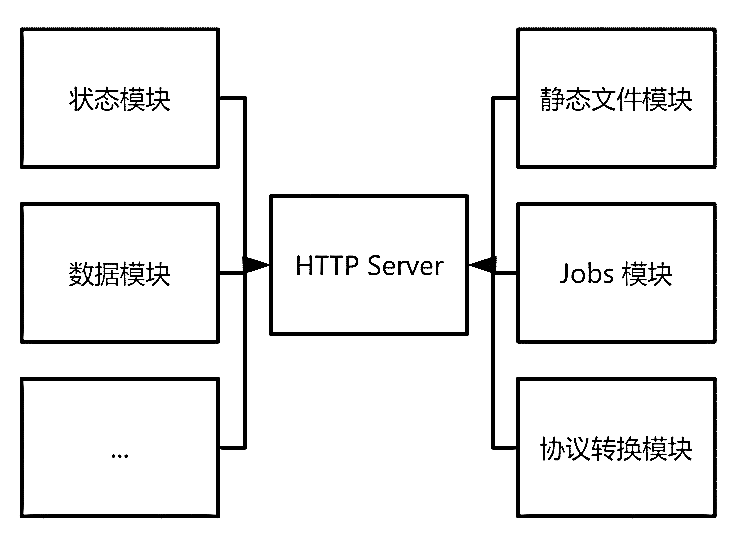


图3.3 HTTP Server模块组成

静态文件模块：提供前端的静态文件

Jobs 模块：提供当前服务器所有任务的相关信息。

协议转换模块：因为 HTTP 协议的缺陷，使用轮询的方式实现双向通信的成本较大，所以需要协议转换模块，使得客户端和服务端之间通信可以转换为 WebSocket。

状态模块：提供当前集群的运行状态。

数据模块：提供多个协议，使得用户可自行将分布式任务的输入数据传入服务器，而不用通知 BMR 系统的管理员。

#### Master

负责传统 MapReduce 系统中，Master 的工作，包括任务调度，监控它们并重新执行失败的任务。使 Worker 按照指示执行任务。

主要由以下几个模块组成：

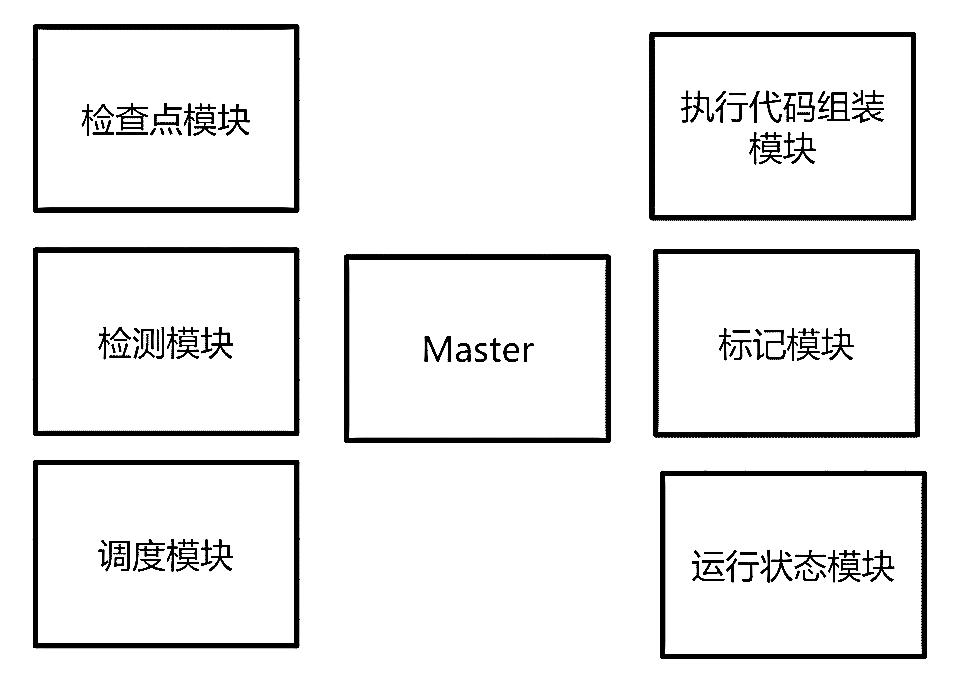


图 3.4 Master模块组成

检查点模块：MapReduce 的架构为单 Master 多 Worker，允许任意多个 Worker 故障，但 Master 不允许故障，如果不做检查点机制，Master 的故障将对分布式计算任务造成巨大的损失。

检测模块：Master 会对所有的 Worker 周期性地发送探活信息，监测 Worker 的工作状态。

调度模块：根据 Worker 加入的 Job 和执行的任务不同，给其发送不同的代码。

执行代码组装模块：将用户编写的 Map 函数和 Reduce 函数，以及插件机制提供的其他用户定义代码，与输入数据，一起封装为一套执行用的代码，发送给 Client 的 Worker。

标记模块：对所有的任务，包括执行中，执行失败，准备执行的代码进行标记，方便调度模块执行重试或者多 Worker 共同执行等机制。

运行状态模块：将 Worker 周期性发送的状态信息进行会中，提供给 HTTP Server，方便用户查看和日志系统进行记录。

### 客户端模块划分

客户端主要分为两部分，一个是传统 MapReduce 系统中，Worker 的部分，另一个则是为用户提供可视化操作和只展示信息的 GUI。客户端中可以有多个 Worker，配合 WEB Workers API，可以充分使用用户设备中的多核处理器。

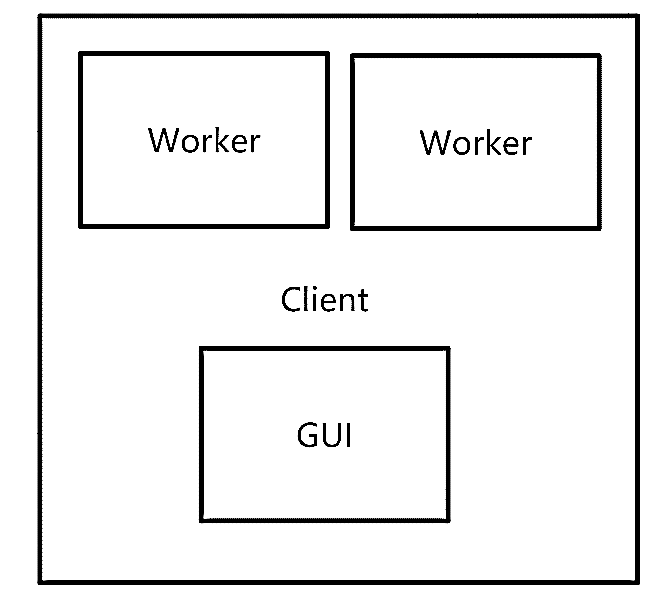
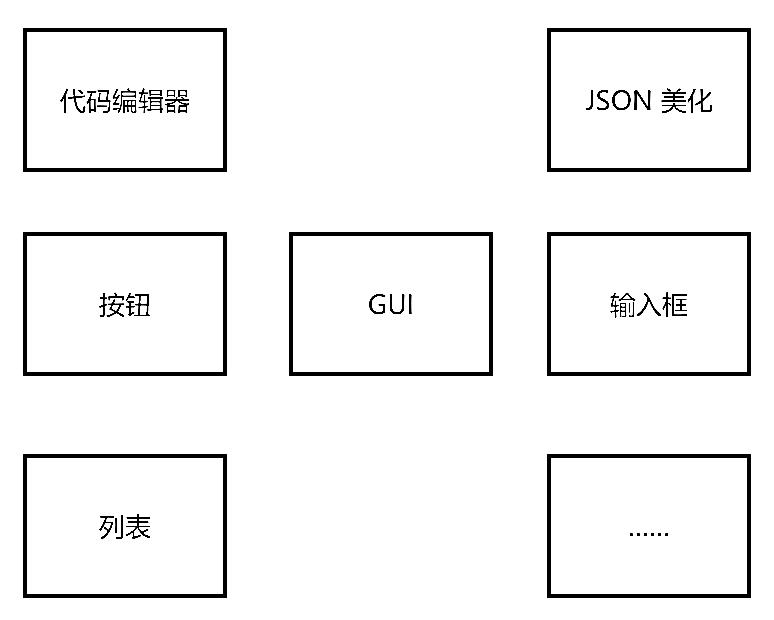


图3.5 客户端组成

#### GUI

GUI 部分，使用 VUE 框架和 ElementUI 组件库开发，使用了组件化的开发方式。这种开发方式强调在整个给定的软件系统中，对广泛的功能进行关注点的分离。它是一种基于重复使用的方法来定义、实现和组成松散耦合的独立组件的系统。这种做法的目的是在短期和长期内为软件本身和赞助这种软件的组织带来同样广泛的利益。



3.6 GUI 模块组成

主要由以下几个组件组成

代码编辑器：用户编写自定义代码，提供了自动补全，高亮等常用功能。

按钮、输入框：基于 ElementUI 的组件库进行二次封装，以达到 BRM 系统的整体设计风格统一。

JSON美化：自封装组件，因为显示分布式任务的执行状态和集群运行状态时，拿到的数据都是 JSON，自封装了一个利用 JSON 自动生成图表的组件。

列表：自封装组件，根据获取的所有 Jobs，自动生成带分页的列表。

### 服务端客户端通信接口

接口分为两部分，分为 HTTP 协议的接口和 WebSocket 的协议。

#### HTTP接口协议

获取 Jobs

HTTP 方法：GET

URL：/job

响应：

|  |
| --- |
| [  {  "name": "word\_counter",  "id": "54991c95-89c0-4aaa-8fb6-90084b1793cf",  "status": "Running",  "workers\_count": 10  },  {  "name": "regular\_search",  "id": "5a9064ea-f75a-4136-bf32-64ed11818efa",  "status": "Running",  "workers\_count": 7  },  {  "name": "arr\_sort",  "id": "54991c95-f75a-4136-bf32-90084b1793cf",  "status": "Running",  "workers\_count": 30  },  ] |

创建 job

HTTP 方法：POST

URL：/job

用户携带数据：

|  |
| --- |
| {  "name": "word\_counter",  "code": {  "map": "function \_map(key, value, data) {data.push(...key.split(' '))}",  "reduce": "function \_reduce(key, values, data) {data.push({[key]: values.length})}"  }  } |

响应：

|  |
| --- |
| {  "name": "word\_counter",  "id": "54991c95-89c0-4aaa-8fb6-90084b1793cf",  "status": "running",  "code": {  "map": "function \_map(key, value, data) {data.push(...key.split(' '))}",  "reduce": "function \_reduce(key, values, data) {data.push({[key]: values.length})}"  },  ...  } |

上传分布式任务所需要的输入数据

HTTP 方法：POST

URL：/data/:job\_id

用户携带数据：格式为 form\_data，并带一个或多个文件

响应：

|  |
| --- |
| [  "54991c95-89c0-4aaa-8fb6-90084b1793cf\_1.slice",  "54991c95-89c0-4aaa-8fb6-90084b1793cf\_2.slice",  "54991c95-89c0-4aaa-8fb6-90084b1793cf\_3.slice",  ] |

获取加入某个 job 的 uuid

HTTP 方法：POST

URL: /gen\_uuid/job\_id

响应：

|  |
| --- |
| "54991c95-89c0-4aaa-bf32-64ed11818efa" |

#### WebSocket协议

WebSocket 是基于消息的，所以格式与 HTTP 有不同

消息名称：“ping”

含义：服务端和客户端的探活信息

消息名称：“status”

含义：服务端和客户端之间的状态信息（如完成记速度，异常情况等）

## 本章小结

本章对 BMR 系统的整体设计进行了说明，总体架构，前后端模块的划分，以及前后端通信所约定的接口。

# 系统实现

模块与模块之间，服务端与客户端之间都使用 json 进行传输数据，一种开放的标准文件格式和数据交换格式，使用人类可读的文本来存储和传输由属性-值对和数组（或其他可序列化的值）组成的数据对象。它是一种非常常见的数据格式，有各种各样的应用，其中一个例子是与服务器通信的网络应用。它源于JavaScript，但许多现代编程语言包括生成和解析JSON格式数据的代码。

## 执行代码组建模块

该模块的输入有三部分，用户写的代码，输入数据，服务器自带的胶水代码，算法逻辑如下：

|  |
| --- |
| function \_exec(user\_obj, input\_data, plugins) {  const config = JSON.parse(plugins['config']);  const user\_function\_name = config['user\_function\_name'];  const user\_function\_code = user\_obj[user\_function\_name];  const user\_function = new Function(`return ${user\_function\_name}`)();  const data = [];  for (const key in input\_data) {  const value = input\_data[key];  user\_function(key, value, data);  }    return data;  }  function \_combine(user\_obj, input\_data, plugins) {  return {  '\_exec': \_exec.prototype.toString(),  'user\_obj': user\_obj,  'input\_data': input\_data,  'plugins': plugins  };  } |

该模块主要由两个函数组成，调用者只需关注 "\_combine" 函数即可。它以用户代码（表现为 user\_obj），此次计算所需数据，和服务器插件对象为输入，输出为一个 json 格式的对象，该对象直接传输给客户端的 Worker，Worker 可根据该对象，反序列化生成执行代码。

## 标记模块

该模块负责记录所有 Task 的完成情况，并根据检测模块传回的 Worker 工作情况，及时更新 Task 的标记。他的功能抽象出来，是维护一个如下列代码格式所示的对象。

|  |
| --- |
| {  "54991c95-89c0-4aaa-8fb6-90084b1793cf": {  'job\_status': 'Running Map',  'files\_status': {  '54991c95-89c0-4aaa-8fb6-90084b1793cf\_1.slice': {  'file\_type': 'map\_input',  'status': 'finished'  },  '54991c95-89c0-4aaa-8fb6-90084b1793cf\_2.slice': {  'type': 'map\_input',  'status': 'running'  },  '54991c95-89c0-4aaa-8fb6-90084b1793cf\_3.slice': {  'type': 'map\_input',  'status': 'failed'  },  '54991c95-89c0-4aaa-8fb6-90084b1793cf\_3.slice': {  'type': 'map\_output',  'status': 'prepared'  },  }  },  "549abc95-81c0-4cdq-8fb6-90084b1793cf": {  'job\_status': 'Running Reduce',  'files\_status': {  '54991c95-89c0-4aaa-8fb6-90084b1793cf\_1.slice': {  'file\_type': 'map\_input',  'status': 'finished'  },  '54991c95-89c0-4aaa-8fb6-90084b1793cf\_2.slice': {  'type': 'map\_input',  'status': 'prepared'  },  '54991c95-89c0-4aaa-8fb6-90084b1793cf\_5.slice': {  'type': 'reduce\_input',  'status': 'failed'  },  '54991c95-89c0-4aaa-8fb6-90084b1793cf\_7.slice': {  'type': 'reduce\_input',  'status': 'prepared'  },  }  },  } |

对外提供两个接口，如代码所示

|  |
| --- |
| function update\_task\_status(job\_id, file\_name, new\_status);  function add\_task\_file(job\_id, file\_name, file\_status); |

其他模块，如检测模块，根据 Worker 的工作情况，调用 update\_task\_status 对文件的完成状态进行更新；或者调用 add\_task\_file，记录产出的中间文件的状态，方便调度模块进行合理的调度。

## 运行状态模块

该模块负责跟踪所有集群运行状态，他的功能抽象出来，是维护以个如下列代码格式所示的对象。

|  |
| --- |
| {  workers: [  {  'worker\_id': '54991c95-89c0-4aaa-8fb6-90084b1793cf',  'job\_id': '54913c95-89c0-4aaa-8fb6-90084b1793cf'  },  {  'worker\_id': 'asdf21-89c0-4aaa-8fb6-90084b1793cf',  'job\_id': '54913c95-89c0-4aaa-8fb6-90084b1793cf'  },  {  'worker\_id': 'ak731jl-89c0-4aaa-8fb6-90084b1793cf',  'job\_id': '54913c95-89c0-4aaa-8fb6-90084b1793cf'  }  ],  'server': {  'server\_name': 'bj\_server\_2c8g',  'server\_ip': '180.76.226.207',  'checkpoint\_time': 1623094629256,  'cpu\_usage': [  0.3, 0.7, 0.6, 0.8  ],  'mem\_usage': 539.8  }  } |

其中 workers 记录了所有在最近一次检测中，仍然存活的的 Worker，及其正在运行的 job 的 id。server 中的 “server\_ip”，"cpu\_usage", "mem\_usage" 会定期更新，保证之后组建更大集群是，提供给上层调度器相应的信息。”checkpoint\_time“ 则记录了最近一次 master 做检查点的日期，方便 master 崩溃后，以最快的速度恢复计算任务。

对外提供修改的接口，但常用的只有修改 ”checkpoint\_time“，其他信息均为自动生成的。

|  |
| --- |
| function update\_checkpoint\_time(time); |

## 检查点模块

该模块负责定期收集计算任务运行状态，生成检查点，方便 master 崩溃后，能及时恢复。核心逻辑如下。

|  |
| --- |
| async function gen\_checkpoint(server\_status) {  const t = Date.now();  await fs.writeFile(`./${t}.checkpoint`, JSON.stringify(server\_status));  emit('update checkpoint', t);  } |

即将传入 server\_statsu 对象，将其反序列化至以时间为文件名的文件中，再触发一个 “update checkpoint” 事件，方便其他模块进行相应操作。

## 检测模块

该模块负责每隔 1s，通过 WebSocket 协议建立的链接，发送“Ping”消息，Worker 发送“Pong”消息，证明自己正常工作中。核心逻辑如下。

|  |
| --- |
| function detect\_workers(workers) {  workers.forEach(async (conn) => {  try {  const resp = await conn.send('ping');  } catch (ex) {  emit('conn\_lost', conn.get\_uid());  continue;  }  emit('update\_status', resp);  });  } |

即传入当前所有已建立好的连接，对每个连接发送 ”ping“ 消息，将未收到的成功消息的 Worker 剔除连接列表中。而对成功的 Worker，更新其工作状态。

## 客户端

### JSON美化组件

由于需要根据服务端传回的，表示运行状态的 JSON 数据，用图表或者文字的方式，展现出来。而且表示状态的数据虽然多，但类型较少。为避免重复工作，所以有了这个组件。这个组件主要负责，根据传回的 JSON 数据，和模板数据，自动生成页面，代码逻辑如下。

|  |
| --- |
| <template>  <div class="json\_viewer">  <div v-for="item in data">  <div v-if="item.type==='text'">  <bmr-text :data="item.data" :pattern="patterns[item.type]"></bmr-text>  </div>  <div v-if="item.type==='graph\_pie'">  <bmr-text :data="item.data" :pattern="patterns[item.type]"></bmr-text>  </div>  <!-- ... -->  </div>  </div>  </template> |

利用 VUE 框架提供的循环功能，自动生成需要渲染数据的每一项。再根据每一项 type，使用不同的自封装组件，并传入数据（item.data）和渲染模板（patterns[item.type]）。这样，这个组件就可被放入客户端的任意位置，只要传入从服务器拿到的状态数据，即可使用可视化的方式向用户展现计算状态或集群运行状态。

### Worker

类似于传统 MapReduce 系统中的 Worker，但再 BMR 系统中，对其做了一定的简化。代码逻辑如下。

|  |
| --- |
| \_exec(exec\_obj) {  const fun = new Function(`return ${exec\_obj['\_exec']}`)();  return fun(exec\_obj['user\_obj'], exec\_obj['input\_data'], exec\_obj['plugins']);  } |

即利用 WEB Workers API，将任务负载从主线程中移到其他线程。执行结束后的结果返回给主线程，交由其返还给 Master。

## 本章小结

本章描述了整个 BMR 系统的实现，对各个模块的功能进行了详细说明，并对系统中，真实运行数据的结构进行了展示，方便更好的理解系统如此实现的原因。

# 系统的使用说明

## 运行环境

### 服务端

首先需要确保服务器内，有 Node.js 的运行环境和 NPM 包管理器。之后使用指令 npm i 安装所需依赖，随后使用 npm run serve 运行服务端。

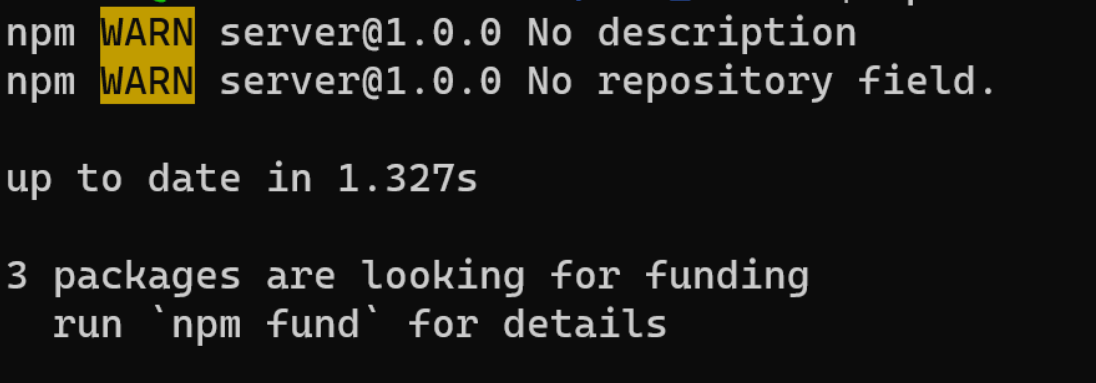


图4.1 安装依赖图

如果所示，为执行 `npm i` 之后，成功的字样。

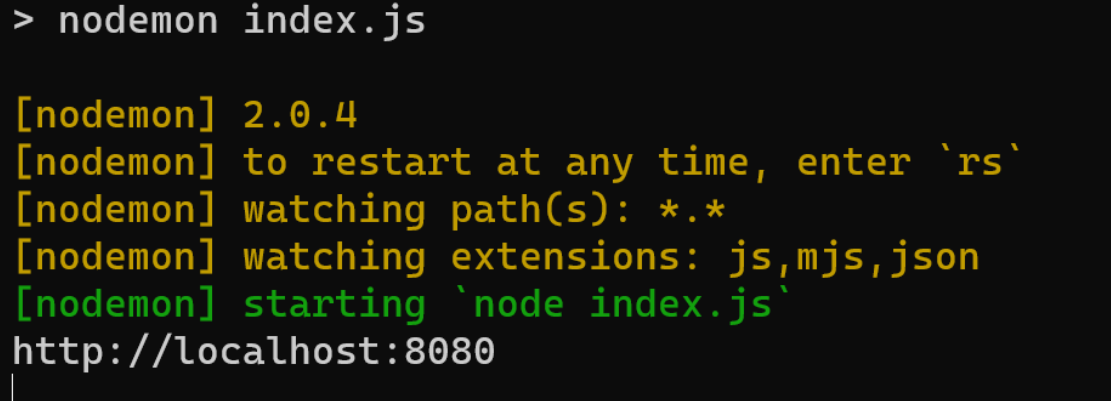


图4.2 服务端运行成功截图

如图所示，成功出现 `http://localhost:8080` 字样，即说明服务端运行成功。

### 客户端

一般直接打开浏览器，访问 `http://localhost:8080` 即可访问前端页面，但为调试方便，有时可能单独启动一个 WEB 服务，运行在其他端口上。因为服务端是以提供静态文件的方式，使得用户访问 `http://localhost:8080` 即可使用。但这次 BMR 系统的开发，采用的是目前比较流行的，以 Vue-cli 提供的脚手架进行开发的方式。源代码并不能直接通过提供静态文件方式提供服务，需要经 Webpack 和 babel 编译之后才行。但开发时，因为开发效率的原因，修改完代码并不会重新编译并部署到服务端中，而是采用脚手架提供的部署工具进行临时的访问。

使用方式还是与服务端类似，先使用 `npm i` 命令，安装相关的依赖。成功后会出现如图所示字样。

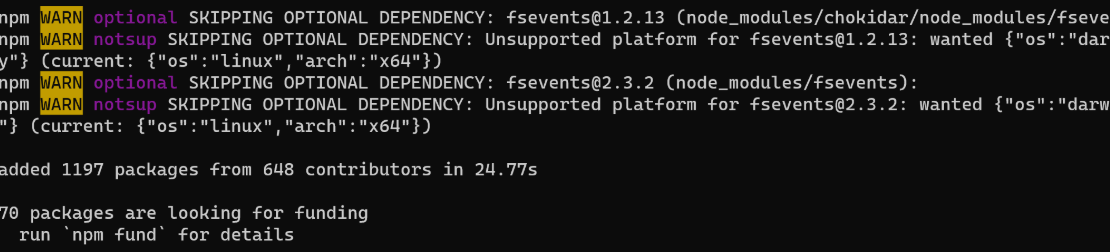


图4.3 安装依赖图

随后使用 `npm run serve` 启动前端服务。成功后会出现如图所示字样。

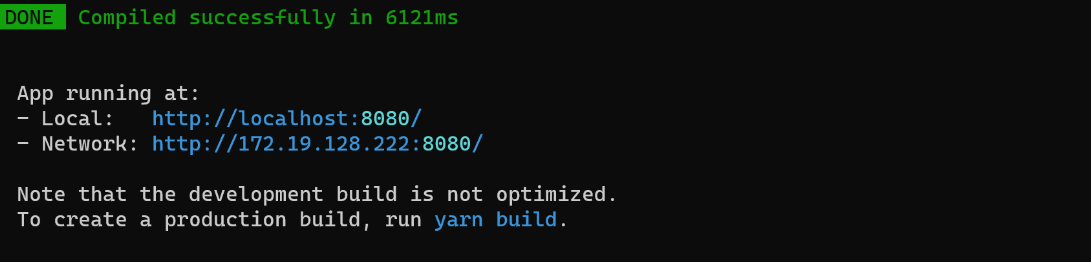


图4.4 前端服务运行成功图

## 源码结构

### 服务端

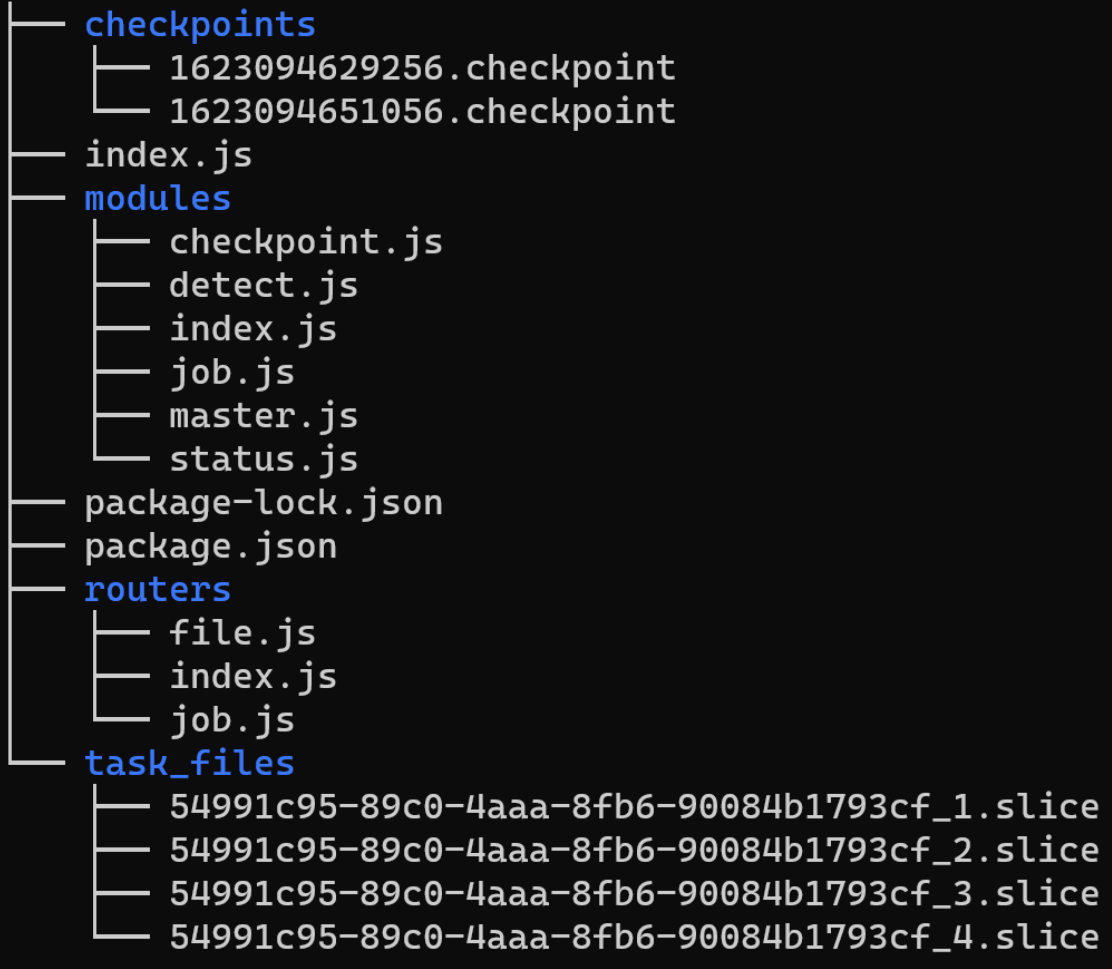


图4.5 服务端架构

checkpoints：存储持久化后的检查点文件

index.js：整个系统的入口文件

modules: 其他模块，index.js 作为整个文件夹的导出文件

router.js：存储着作为 HTTP Server 的相关接口文件，index.js 作为整个文件夹的导出文件

task\_files：存储着分布式计算任务的输入及输出文件

package.json 和 package-lock.json：系统用到的所有依赖及版本

### 客户端

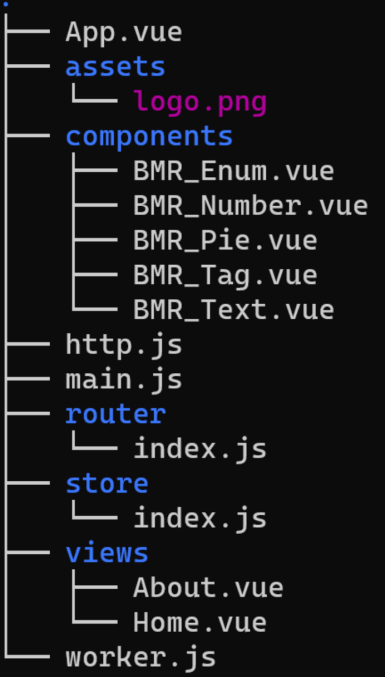


图4.6 客户端架构

App.vue：前端最顶层的页面

main.js：加载页面加载的入口代码

http.js：基于 axios 自行封装的 http 库

worker.js：worker 的主要逻辑，负责从服务器接收任务，利用 WEB Workers API 生成新线程，协议转换等功能。

assets：存放使用到的静态图片等资源。

components：详细设计提到的自封装组件，方便对代码的复用。

router：前端的路由相关逻辑。

store：状态管理相关，因为 BMR 系统涉及到的状态转换相对复杂，不使用全局的状态管理，将很难掌控整体的复杂性。

## 使用说明

使用者无需关注服务端，上传完成数据文件后，即可在代码编辑器组件中编写自定义函数。之后等待运行完成即可。下面是示例代码。

|  |
| --- |
| function \_user\_code() {  return {  '\_map': (key, value, data) => {  const arr = value.split(' ');  for (const idx in arr) {  const word = arr[idx];  data.push([key, 1]);  }  },  '\_combinator': (key, values, data) => {  const dict = {};  for (const idx in values) {  const value = values[idx];  dict[value] = dict[value] ?? 0;  dict[value] += 1;  }  for (const word in dict) {  const count = dict[word];  data.push([word, count]);  }  },  '\_reduce': (key, values, data) => {  data.push([key, values.reduce((a, b) => a + b, 0)]);  }  }  } |

这段代码的功能是，统计文本文件的单词出现数量，中途为了降低网络传输的代价，使用了“combinator”的机制，对“map”函数的输出做了提前整合。最终输出为如下列代码所示的 json 对象。

|  |
| --- |
| {  "apple": 1000351,  "banana": 501113  } |

用户只需要定义一个返回一个包含map、reduce函数的对象，即可使用到集群中所有现存和将来加入的 Worker。

因为使用 JavaScript 作为自定义执行代码，即使是对分布式不怎么了解的前端工程师，也可使用集群中的运算资源。又因为 BMR 系统是基于浏览器的，所以当有对时间不敏感，但计算时间较长的任务时。在社交网络上联系好友们，让他们开几个浏览器窗口挂在后台，就可以使用各式设备帮助我们加速计算速度，如手机、平板、甚至是智能电视。

## 本章小结

本章对系统的部署，运行进行了说明，还阐述了整个系统的代码结构。并枚举了用户使用该系统编写分布式计算任务时所需要注意的事项

# 结论

## 论文总结工作

本文提出的基于浏览器的 MapReduce 分布式计算系统，使更多的开发人员也能享受到 MapReduce 带来的简洁优雅的开发体验。而基于浏览器的设计，使得在急需算力时，在社交网络求助三两好友，在电脑甚至是手机平板上，开几个标签即可加入集群。

利用大数据技术分析海量数据逐渐成为IT行业的热门话题，越来越多的公司相信大数据技术将带领他们走向成功。本文提出了一个基于浏览器的MapReduce分布式计算系统，使普通开发者也能享受到MapReduce带来的优雅、快速的分布式计算体验。基于浏览器的设计使得在急需计算能力的时候，可以向社交网络中的三两个朋友求助，在笔记本电脑甚至是平板电脑、手机上打开几个标签就可以加入集群。

集群的加入，Master 的部署，MapReduce 任务的编写这三个部分的实现，是论文详细介绍了的。

集群的加入，即浏览器成为传统 MapReduce 系统中，Worker 的部分，是 BMR 系统的关键部分之一。该模块是以用户的浏览器作为平台，配合现代的，全双工式的通信协议和 WEB Workers API充分利用宿主设备的高速多核心现代处理器。同时也使得单线程，事件循环驱动的 JS，得以正常运行。即使在运行高负载的 Map 和 Reduce 函数时，也不阻塞页面，用户得以正常且流程的操作页面。

Master 的部署，即服务端作为传统 MapReduce 系统中，Master 的部分。该模块以 Node.js 作为运行环境，配合 JS 的灵活性，可在服务端组合好代码，配以需要处理的数据，打包发给前端。

MapReduce 任务的编写，即代替传统 MapReduce 系统中，上传的用户进程。因为 JS 在浏览器中即可运行，所以用 JS 作为 BMR 系统中，用户自定义函数所需要编写的代码的编程语言，也十分自然。同时因为 JS 可在浏览器中运行的特性，可以在网页上给用户提供代码高亮，智能提示等功能，使得用户在 BMR 系统中的工作流，十分顺畅。

以上就是 BMR 系统的主要功能模块，系统在经过测试后，功能和性能上都基本完成预期的目标。

## 未来展望

BMR系统在一些方面仍然需要提高：

1. 性能的缺陷。JS 代码虽然在 Google 的 V8 引擎的优化下，效率已有很大的提升，但对于传统的静态类型语言，如 C++，JAVA 等，仍有 4-5 倍的差距。之后可以考虑使用 WASM 等技术，使得用户可以编写 C++、RUST 等代码，运行在 BMR 系统中。还可以尝试加入异构计算，如用户可使用一些库，如 WEB GL，利用用户的 GPU，提升节点的专用计算能力。
2. 库的缺陷。目前 BMR 系统，用户编写的 MapReduce 库中，只能使用 JS 原生的库，对数学运算不太友好。可以考虑加入一个内嵌的包管理工具，使得用户可以在编写代码时，加入预定义的宏，服务端再根据预定义的宏，嵌入用户所需要的包。提升开发效率，降低用户的使用门槛。

此外，任务工作的状态的展现，受限于个人的前端技术，还较为原始，改进的空间较大。可以考虑之后再做美化，改进展现方式，以期给用户带来优雅，直观的状态信息。

# 参考文献

1. 黄宏涛, 王慧. 基于Web的分布式计算[J]. 计算机应用研究, 2000, 017(002):38-41.
2. 张文峰. 基于MapReduce模型的分布式计算平台的原理与设计[D]. 华中科技大学, 2012.
3. 程序员的自我修养：链接、装载与库/俞甲子，石凡，潘爱民著.北京：电子工业出版社，2009.4
4. 计算机网络：自顶向下方法/（美）库罗斯（Kurose，J.F.），（美）罗斯（Roos，K.W.）著；陈鸣译.北京：机械工业出版社，2014.9
5. 申时全.基于Linux多线程技术的网络并发编程及应用研究[J].现代计算机(专业版),2016(31):65-70.
6. Jeffrey, Dean, Sanjay, et al. MapReduce: simplified data processing on large clusters[J]. Communications of the Acm, 2008.
7. Ghemawat S , Gobioff H , Leung S T . The Google File System[C]// ACM. ACM, 2003:29.
8. Chang F , Dean J , Ghemawat S , et al. distributed storage system for structured data. 2008.
9. 李建江, 崔健, 王聃,等. MapReduce并行编程模型研究综述[J]. 电子学报, 2011, 39(11):2635-2642.
10. Tilkov, Stefan, Vinoski, et al. Node.js: Using JavaScript to Build High-Performance Network Programs[J]. IEEE Internet Computing, 2010.
11. 王金龙, 宋斌, 丁锐. Node.js:一种新的Web应用构建技术[J]. 现代电子技术, 2015(06):70-73.
12. 许会元, 何利力. NodeJS的异步非阻塞I/O研究[C]// 2015:127-129.
13. 程桂花, 沈炜, 何松林,等. Node.js中Express框架路由机制的研究[J]. 工业控制计算机, 2016, 29(8):101-102.
14. 薛陇彬, 刘钊远. 基于WebSocket的网络实时通信[J]. 计算机与数字工程，2014(03):478-481.
15. 孙寅林. 基于分布式计算平台的海量日志分析系统的设计与实现[D]. 西安电子科技大学.
16. 冯萍慧, 连一峰, 戴英侠,等. 基于可靠性理论的分布式系统脆弱性模型[J]. 软件学报, 2006, 017(007):1633-1640.
17. 陶雪娇, 胡晓峰, 刘洋. 大数据研究综述[J]. 系统仿真学报, 2013(S1):142-146.
18. 刘旭. ChromeV8引擎中的JavaScript数组实现分析与性能优化[J]. 计算机与现代化, 2014(10):66-70.
19. 张迎周, 张卫丰. Haskell:一种现代纯函数式语言[J]. 南京邮电大学学报(自然科学版), 2007, 27(004):13-18.
20. 朱二华. 基于Vue.js的Web前端应用研究[J]. 科技与创新, 2017, 000(020):119-121.

# 致谢

岁月如梭，入学时学校的景色还历历在目，毕设完成后，回学校看了看曾经走过无数次的路，和怡园——常去的食堂。吃饭时看到匆忙赶去课堂的学弟学妹们，突然意识到，所有故事都会有一个结尾。我还有许多专业知识想学习，还有许多有趣的项目没有做，还想和老师们探讨技术选型，与同学们畅想未来。

首先感谢学校，从小生活在南方的我，收到录取通知书以后，会觉得无法适应北方的生活习惯，不适应这边的气候。学校对我们的关心，食堂丰富的菜品，使得我们以极快的速度，适应了这边的生活。同时也感谢学校大力支持各类科创比赛，我们在此期间获得了宝贵的参赛经验，提升了自己的专业素养和也获得了面对压力时冷静的心态。

其次感谢老师，在高中时期的自己，即使是上讲台，也会感到紧张。这四年间，不管是班导，专业课老师还是社团指导老师，都给了我了许多锻炼的机会。自己的社交技能也在这一次次的锻炼中，获得了提高。十分感谢老师送给自己这份珍贵的礼物。

还要感谢同学，方俊、贾利凯作为大学时期中认识的好友，在我比赛失利时给我的安慰，在我压力大给我的鼓励。也要感谢在社团中认识的姜威学长、王建力学长，他们时常在群里分享业界目前的热点方向，指导我们的学习路线。也在繁忙的工作中，抽出时间维护社团的官网，使我们这期间可以将自己脑中的想法，转换为真正的代码，并上传至 OJ，交由判题机进行自动判断，实时反馈结果。这对我之后的工程能力有很大的提高。

最后感谢我的指导老师，刘新老师。在我实现 BMR 系统时，悉心的指导，使得整个实现过程中，都是按照计划顺利进行，整个过程中遇到的难题，也恰好时自己努努力就能解决的难度。论文的书写也讲解了书写方面的技巧，让我得以成功完成自己的毕业设计论文。