

分布式任务执行框架 MARC 说明文档

丁国栋¹

中国科学院计算技术研究所

Last-Modified: 2011 年 7 月 29 日

目 录

| | |
|------------------------------------|----|
| 分布式任务执行框架 MARC 说明文档..... | 1 |
| 一、架构及功能简介..... | 2 |
| 二、系统特性..... | 5 |
| 三、系统流程..... | 7 |
| 3.1 Master 节点运行流程..... | 7 |
| 3.2 Result 节点运行流程..... | 7 |
| 3.3 Client 节点运行流程..... | 7 |
| 3.4 Client 节点与 Master 节点的交互过程..... | 7 |
| 3.5 Result 节点与 Master 节点的交互过程..... | 7 |
| 3.6 Client 节点与 Result 节点的交互过程..... | 7 |
| 四、接口说明..... | 7 |
| 4.1 任务生成应用程序接口说明..... | 8 |
| 4.2 任务执行应用程序接口说明..... | 9 |
| 4.3 结果处理应用程序接口说明..... | 9 |
| 五、安装部署说明..... | 10 |
| 5.1 编译及安装(以 Linux 为例)..... | 10 |
| 5.2 Master 节点运行及配置文件简要说明..... | 11 |
| 5.3 Result 节点运行及配置文件简要说明..... | 15 |
| 5.4 Client 节点运行及配置文件简要说明..... | 18 |
| 六、前台管理界面..... | 21 |
| 6.1 首页..... | 22 |
| 6.2 Master 节点信息..... | 22 |
| 6.3 Result 节点信息..... | 23 |
| 6.4 Client 节点信息..... | 23 |
| 6.5 任务执行状态..... | 24 |
| 6.6 Master 日志信息..... | 24 |
| 6.7 Master 异常日志信息..... | 25 |
| 附录 1 系统开发简介..... | 26 |
| 附录 2. 2011 年 7 月升级与完善列表..... | 28 |
| 附录 3 待开发的功能及完善列表..... | 29 |

¹ 作者联系方式: 丁国栋, dingguodong@ict.ac.cn 或 gdding@hotmail.com

一、架构及功能简介

MARC 是一个分布式的任务调度与执行框架，由三个部分组成：Master 节点、Result 节点和 Client 节点。MARC 中的几个字母分别表示：M - Master, A - Application, R - Resut, C - Client。下图为 MARC 的系统架构图。

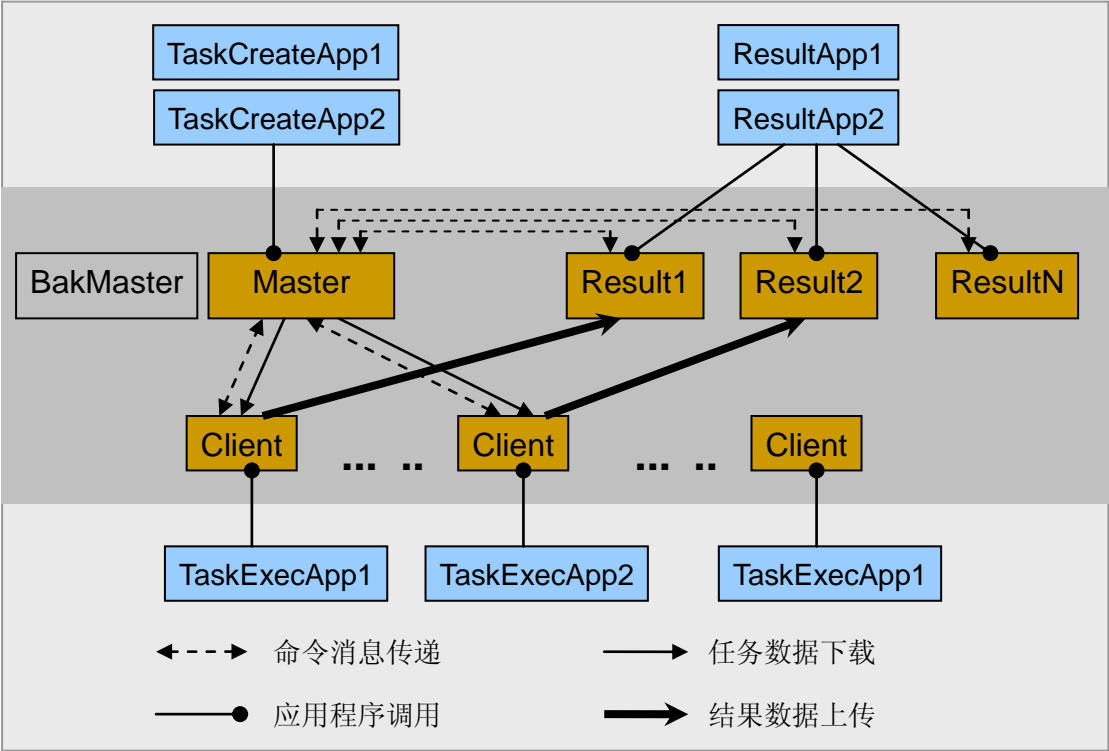


图 1. MARC 系统架构

上图中，中间灰色背景部分即为 MARC 框架，包括 Master 节点（及备用 Master 节点）、一个或多个 Result 节点、以及一个或多个 Client 节点；外围的部分为用户的应用程序，包括部署在 Master 节点上的“任务生成应用程序”如 TaskCreateApp1、部署在 Client 节点上的“任务执行应用程序”如 TaskExecApp1、以及部署在 Result 节点上的“结果处理应用程序”如 ResultApp1。

【Master】

Master 是整个框架的“管理者”，在整个框架中只有一个 Master 节点（可配置一个备用 Master 节点），其职责是：

- (1) 接收各 Client 节点、Result 节点的注册请求和运行状态信息；
- (2) 通过心跳检测机制维护各个 Client 节点、Result 节点的有效性；
- (3) 在 Client 节点需要任务时自动调用用户编写的任务生成应用程序，并将其生成的任务告知 Client 节点以便让 Client 节点自行下载；

- (4) 在 Client 节点任务执行完成需要上传任务执行结果时, Master 会选择负载最小的 Result 节点, 将其存储地址告知 Client 节点;
- (5) 监控和管理各个 Client 节点的任务执行情况, 等等。
- (6) 监控和管理各个 Client 节点和 Result 节点上的应用程序版本, 自动对应用程序升级包进行压缩和管理, 方便各个节点下载和升级。
- (7) 定期收集当前机器的计算资源信息, 如可用磁盘空间大小、内存大小、CPU 等。
- (8) 提供一个内嵌的 HTTP Server, 使得框架的用户或系统运维人员能方便地通过浏览器实时地查看框架中各个节点的运行状态、资源使用情况、任务执行状况、日志信息等。

【Application】

Application 表示用户的应用程序。MARC 系统只是一个任务执行框架, 具体用来执行什么样的任务, 是用户应用程序所赋予的功能 (在一个 MARC 框架中, 可以部署不同功能的应用程序, 即允许各种类型的应用程序共存于同一个 MARC 框架中, 比如新闻采集器、论坛采集器等)。

具体来说, MARC 框架的使用者可按照 MARC 提供的接口规范分别编写部署在 Master 节点上的“任务生成应用程序”、部署在 Client 节点上的“任务执行应用程序”和部署在 Result 节点上的“结果处理应用程序”, 这三个应用程序与 MARC 框架共同组成了一个基于任务的分布式运行系统, 其基本流程是: “任务生成→任务下载→任务执行→结果回传→结果处理”, 具体地:

- ◆ **任务生成:** 在 Master 节点上, MARC 框架会自动监测每个 Client 节点的任务执行状况, 当某个 Client 节点空闲时, MARC 框架会自动调用用户的“任务生成应用程序”, 然后将生成的任务数据压缩后缓存在本地等待 Client 节点下载。
- ◆ **任务下载:** 当 Client 节点从 Master 节点获知其任务生成完成后, Client 节点自动从 Master 节点下载生成的任务数据。为提高传输效率, 任务数据首先在 Master 节点经过压缩处理, Client 节点接收完成后再进行解压。
- ◆ **任务执行:** Client 节点下载完任务后, MARC 框架会自动调用用户的“任务执行应用程序” (以异步、非阻塞的方式执行)。
- ◆ **结果回传:** Client 节点的“任务执行应用程序”运行完成后, MARC 框架会自动将任务执行的结果数据先行压缩后再上传到相应的 Result 节点 (Master 节点会告知 Client 节点其任务执行结果应该上传到哪个 Result 节点)。

- ◆ **结果处理：**Result 节点在接收完 Client 节点上传的任务执行结果后，会自动调用用户的“结果处理应用程序”（调用前先进行解压），完成比如结果入库或进一步的其他处理工作。

以上各个过程之间是可以并行进行的，比如，可以设置为异步上传策略，这样当任务执行完成后，可以马上获取新任务而不必等待结果上传完毕。

【Result】

Result 节点是整个框架的结果收集和处理端，在整个 MARC 框架中可以部署一个或者多个 Result 节点。Result 节点的功能包括：

- （1）负责接收 Client 节点上传的任务执行结果，接收完成后，自动调用相应的“结果处理应用程序”完成对任务执行结果的处理工作。
- （2）定期发送心跳信息和节点运行状态信息给 Master 节点。
- （3）通过与 Master 通信，定期检查本地的应用程序版本是否需要升级，若需要升级则自动下载升级包并进行升级。
- （4）当 Master 节点无法连接时，重启，自动切换到 Master 备用节点。
- （5）定期收集当前机器的计算资源信息，如可用磁盘空间大小、内存大小、CPU 等，发送给 Master 节点。

Result 节点之间相互独立，不会有任何通信。

【Client】

Client 节点是 MARC 框架中的任务执行端，在整个 MARC 框架中可以部署多个 Client 节点。Client 节点的功能包括：

- （1）空闲时向 Master 节点请求任务，得到任务信息后从 Master 节点自动下载任务数据。任务下载到本地后，自动调用相应的“任务执行应用程序”。
- （2）“任务执行应用程序”运行完成后，自动将结果数据回传给相应的 Result 节点。
- （3）定期发送心跳信息和节点运行状态信息给 Master 节点。
- （4）通过与 Master 通信，定期检查本地的应用程序版本是否需要升级，若需要升级则自动下载升级包并进行升级。
- （5）当 Master 节点无法连接时，重启，自动切换到 Master 备用节点。
- （6）定期收集当前机器的计算资源信息，如可用磁盘空间大小、内存大小、CPU 等，发送给 Master 节点。

Client 节点之间相互独立，不会有任何通信。

二、系统特性

■ 通用性

- ✓ 通过抽象和剥离应用程序的接口，使得 MARC 可以独立于具体的应用（Application），成为一个通用的任务执行框架，可以用来执行各种类型的任务。比如，当 Application 为采集应用时，就是分布式采集系统；当 Application 为索引创建应用时，就是分布式的索引创建系统。
- ✓ 用户要使用 MARC 开发某种功能的应用时，只需要根据 MARC 的应用程序接口规范编写独立的“任务生成应用程序”、“任务执行应用程序”和“结果处理应用程序”，然后部署在 MARC 框架上，就可以在数十台或数百台机器上完成大规模分布式的任务处理工作。

■ 可扩展性及鲁棒性

- ✓ 各节点之间通过 TCP/IP 进行通信和数据传输，能轻松地部署在局域网以及广域网甚至 Internet 环境下。
- ✓ 可为 Master 节点部署一个备用节点以解决 Master 的单点故障问题。
- ✓ Result 节点可动态扩展（数量不限），使整体的结果回收处理能力得到线性增强。在系统运行期间，可动态加入新的 Result 节点或删除旧的 Result 节点而不影响整个框架的正常运行。
- ✓ Client 节点可动态扩展（数量不限），使整体的任务执行能力得到线性增强。在系统运行期间，可动态加入新的 Client 节点或删除旧的 Client 节点而不影响整个框架的正常运行。

■ 自适应性

- ✓ Master 节点出现故障时，Client 节点/Result 节点会自动切换到 Master 备用节点
- ✓ Master 节点通过心跳检测机制自动感知 Client 节点以及 Result 节点的有效性
- ✓ Master 节点自动检测 Result 节点负载情况，并据此选择最优节点告知 Client 节点
- ✓ 故障节点任务的自动调整：当某个 Client 节点出现故障而无法完成任务时，自动将之前该节点未完成的任务分配给其他 Client 节点
- ✓ Master 会自动收集和管理 Client 节点以及 Result 节点的计算资源信息
- ✓ 自动对各个节点上的应用程序版本进行升级和管理

■ 易用性

- ✓ 应用开发简单：MARC 框架的接口十分简单，要使用 MARC 框架，应用程序开发者仅需要开发满足 MARC 应用程序接口规范的三个应用程序即可：“任务生成应用程序”、“任务执行应用程序”以及“结果处理应用程序”。
- ✓ 部署维护容易：通过 MARC 框架各个节点对应的配置文件如 Master 节点上的 master.ini 文件、Result 节点上的 result.ini 文件以及 Client 节点上的 client.ini 文件，仅需要简单的配置就可以完成整个系统的部署工作，在系统长期运行过程中，MARC 框架不需要耗费多少系统资源即可长期稳定运行，此外，各个节点上还有十分详细易读的日志文件供系统管理员进行查看和分析。
- ✓ 应用程序的升级容易：可以将各种应用程序类型（AppType）的升级包统一放在 Master 节点上，MARC 框架就会自动对各个节点上的应用程序进行版本检查和自动升级。

■ 安全性

- ✓ 在系统初始化阶段，Client 节点以及 Result 节点启动时，必须向 Master 注册，Master 会对其进行合法性验证，从而有效避免非法的节点接入 MARC 系统。
- ✓ 在系统长期运行阶段，Client 节点与 Master 节点之间、Client 节点与 Result 节点之间、Result 节点与 Master 节点之间的所有通信消息以及传输的数据等都经过加密处理，充分确保节点之间传输的数据安全性。

■ 跨平台支持

- ✓ 使用标准的 C/C++ 开发，能运行在各种 Windows 系统、Linux 系统以及 Unix 系统上。依赖的第三方库包括 zlib（用于数据压缩及解压）和 libevent（用于提供内嵌的 HTTP Server）。
- ✓ 在一个实际部署的 MARC 框架中，各个节点可部署在不同的硬件平台、操作系统平台上，如 Master 部署在 Linux 上、Client 部署在 Windows 上等。

三、系统流程

3.1 Master 节点运行流程

略。待补充。

3.2 Result 节点运行流程

略。待补充。

3.3 Client 节点运行流程

略。待补充。

3.4 Client 节点与 Master 节点的交互过程

略。待补充。

3.5 Result 节点与 Master 节点的交互过程

略。待补充。

3.6 Client 节点与 Result 节点的交互过程

略。待补充。

四、接口说明

前面已提到，MARC 只是一个通用的任务执行框架，用来解决针对大规模任务并发执行时需要多个节点来执行任务的需求，具体用来执行什么样的任务，是用户应用程序 Application 所赋予的功能。要使用 MARC 来执行某种类型的任务，就需要针对这种类型的任务开发三个相应的应用程序，即“任务生成应用程序”、“任务执行应用程序”和“结果处理应用程序”，三个应用程序分别由 MARC 框架中的 Master 节点、Client 节点和 Result 节点来调用执行。这三个应用程序的输入输出接口必须满足 MARC 所要求的接口规范，以下就对接口规范进行详细阐述。

由于同一个 MARC 框架中可以支持不同类型的应用程序同时存在和运行，就需要为不同类型的应用程序确定一个唯一的名称 `AppType`，这个名称贯穿于整个框架的配置文件如 `client.ini`、`master.ini` 以及 `result.ini` 中，如给新闻采集应用的 `AppType` 命名为 “`NewsGather`”，给论坛采集应用的 `AppType` 命名为 “`ForumGather`”，给倒排索引创建应用的 `AppType` 命名为 “`IndexBuilder`”，等等。

4.1 任务生成应用程序接口说明

“任务生成应用程序”是一个可执行程序，部署在 `Master` 节点上，其职责是为给定的 `Client` 节点生成其任务文件。当 `Master` 节点检测到某个 `Client` 节点空闲时，就会通过 `fork`（Windows 下通过 `ShellExecuteEx`）创建一个子进程执行相应的“任务生成应用程序”。

“任务生成应用程序”应该是一个能在有限时间内执行完成的可执行程序，当 `Client` 节点需要任务时，MARC 框架会自动调用相应类型的任务生成程序，调用接口为：`[AppCmd] [TaskDirPath] [ClientID]`，如：

“`./NewsTaskCreate ./task/12_NewsGather_task_20100204095923/ 12`”

其中`[AppCmd]`可看成是任务生成应用程序的名称，后两个参数`[TaskDirPath]`和`[ClientID]`可以看成是任务生成应用程序的命令行参数，在 MARC 框架调用任务生成应用程序时，会自动将这两个参数通过命令行方式传给任务生成应用程序。

- ◆ `[AppCmd]`是在 `Master.ini` 中设置的任务生成程序的执行串，如在 `Master.ini` 中为应用程序类型为“`NewsGather`”的任务生成程序执行串设置为“`./NewsTaskCreate`”，则 `Master` 节点在调用时，将以“`./NewsTaskCreate [TaskDirPath] [ClientID]`”为命令来执行任务生成程序；若设置为“`NewsGather=./TaskCreate 1`”，则将以“`./TaskCreate 1 [TaskDirPath] [ClientID]`”为命令来执行任务生成程序。
- ◆ `[TaskDirPath]`是一个用于存放任务数据文件的路径，该路径由 `Master` 节点自动创建，形如“`./task/1_NewsGather_task_20100204095923/`”。任务生成应用程序应将其创建的任务数据文件存放在该文件夹中，在任务生成应用程序执行完成后，该文件夹会被 `Master` 节点压缩缓存在本地等待相应的 `Client` 节点下载。`Client` 节点下载解压完成后，该文件夹即为 `Client` 节点的 `input` 文件夹，以命令行参数的形式传递给任务执行应用程序。为了让 `Master` 节点知道任务生成应用程序执行成功（即为给定的 `Client` 节点成功创建了任务），须在`[TaskDirPath]`文件夹中生成“`.success`”文件，否则 MARC 框架将不会将其下发给 `Client` 节点。
- ◆ `[ClientID]`是 `Client` 节点的唯一性编号（整数类型）。

4.2 任务执行应用程序接口说明

“任务执行应用程序”是一个可执行程序（须在有限时间内执行完成），部署在 Client 节点上，其职责是为根据给定的任务数据执行任务处理工作。

当 MARC 框架的 Client 节点从 Master 节点上下载完成任务数据并解压到本地后，就会通过 fork（Windows 下通过 ShellExecuteEx）创建一个子进程执行相应的“任务执行应用程序”。调用接口为：`[AppCmd] [InputDir] [OutputDir] [ID]`，如：`./NewsGather.exe ./input/ ./output/ 12`。其中`[AppCmd]`可看成是“任务执行应用程序”的名称，后三个参数`[InputDir]`、`[OutputDir]`和`[ClientID]`可以看成是“任务执行应用程序”的命令行参数，在 MARC 框架调用“任务执行应用程序”时，会自动将这三个参数通过命令行方式传给“任务执行应用程序”。

- ◆ `[AppCmd]`是在 `client.ini` 中设置的任务执行应用程序的执行串，如在 `client.ini` 中将 `AppType` 设置为“NewsGather”，`AppCmd` 设置为“`AppCmd=./NewsGather`”，则表示该 Client 节点的任务执行应用程序的类型为“NewsGather”，其任务执行应用程序的执行串为“`./NewsGather`”，则 Client 节点在调用时，将以“`./ NewsGather [InputDir] [OutputDir] [ClientID]`”为命令来执行任务生成程序；若 `AppCmd` 设置为“`AppCmd=./Gather 1`”，则将以“`./Gather 1 [InputDir] [OutputDir] [ClientID]`”为命令来执行任务执行应用程序。
- ◆ `[InputDir]` 是一个文件夹路径，该文件夹用于存放任务执行应用程序的输入数据，对应于 MARC 框架的 Master 节点的任务生成应用程序创建的任务生成结果文件夹即上节中的`[TaskDirPath]`。
- ◆ `[OutputDir]`是一个文件夹路径，该文件夹用于存放任务执行应用程序的执行结果数据。任务执行应用程序应将其执行结果文件存放在该文件夹中，在执行完成后，该文件夹会被 Client 节点自动回传到一个 Result 节点上。用户应用程序执行成功后，须在 MARC 框架的当前目录下生成一个标志文件“`.success`”，以向 MARC 框架表明程序执行成功（否则 MARC 框架将不会将其执行结果回传）。
- ◆ `[ClientID]`是 Client 节点的唯一性编号（整数类型）。

4.3 结果处理应用程序接口说明

“结果处理应用程序”是一个可执行程序（须在有限时间内执行完成），部署在 Result 节点上，用于对任务执行的结果数据进行相关的处理工作（如入库或

其他的进一步处理)。

当 Result 节点接收到某个 Client 节点回传的任务执行结果解压到本地后,就会通过 fork (Windows 下通过 ShellExecuteEx) 创建一个子进程执行相应的“结果处理应用程序”。调用接口为: [AppCmd] [ResultPath] [ClientID], 如: `"/.NewsStore ./data/12_NewsGather_result_20100204095923/ 12"`。其中[AppCmd]可看成是“结果处理应用程序”的名称,后两个参数[ResultPath]、[ClientID]可以看成是“结果处理应用程序”的命令行参数,在 MARC 框架调用“结果处理应用程序”时,会自动将这两个参数通过命令行方式传给“任务执行应用程序”。

- ◆ [AppCmd]是在 result.ini 中设置的结果处理应用程序的执行串,如在 result.ini 中为应用程序类型为"NewsGather"的任务生成程序执行串设置为“./NewsStore”,则 result 节点在调用时,将以“./ NewsStore [ResultPath] [ClientID]”为命令来结果处理生成程序;若设置为“./Store 1”,则将以“./Store 1 [ResultPath] [ClientID]”为命令来结果处理应用程序。
- ◆ [ResultPath] 是待处理结果的文件夹路径,其存放的内容即是任务执行应用程序的执行结果(即上节中的[OutputDir])。
- ◆ [ClientID]是 Client 节点的唯一性编号(整数类型)。

五、安装部署说明

5.1 编译及安装(以 Linux 为例)

(1) 通过 svn 下载 MARC 最新源码(最新源码地址请咨询作者):

```
svn export http://10.61.1.244/svn/ZJH/src/MARC/trunk MARC
```

(2) 进入到 MARC 目录后,执行 rebuild.sh,即进行编译。

(3) 编译完成后,Master 节点的 marc 程序及其他相关文件位于 bin/marc_master, Result 节点的 marc 程序及其他相关文件位于 bin/marc_result, Client 节点的 marc 程序及其他相关文件位于 bin/marc_client,这三个文件夹中分别还有三个用于测试 MARC 框架的任务生成应用程序 TaskCreateTest.sh (Windows 下为 TaskCreateTest.bat)、结果处理应用程序 ResultStoreTest.sh (Windows 下为 ResultStoreTest.bat)以及任务执行应用程序 Test.sh (Windows 下为 Test.bat)。

(4) 用户若要运行自己的应用程序,最好将应用程序放置于 MARC 的同级目录下,比如将任务生成应用程序“NewsTaskCreate”放在 marc_master 文件夹中(理论上不在一个文件夹中也可以,但没有测试过这种情况)。

5.2 Master 节点运行及配置文件简要说明

【Master 节点程序目录结构】

```
[gdding@localhost marc_master]$ ll
total 2136
-rwxr-xr-x 1 gdding gdding 1737513 May 31 13:57 marc
-rw-rw-r-- 1 gdding gdding 5036 May 22 21:53 master.ini
-rwxr-xr-x 1 gdding gdding 423243 May 31 13:57 myzip
-rwxrwxr-x 1 gdding gdding 129 Mar 19 09:46 TaskCreateTest.sh
```

其中 marc 文件即是 MARC 框架的主程序, master.ini 为 Master 节点的配置文件, myzip 为 MARC 框架中使用的内部压缩与解压工具, TaskCreateTest.sh 为 Linux 下用于测试的一个任务生成应用程序。

【Master 节点启动命令】

在 marc_master 目录下以 master 方式执行 marc 程序即可。如下图所示。

```
[gdding@localhost marc_master]$ ./marc master
<2010-05-31 14:02:49> Info: 准备启动Master...
<2010-05-31 14:02:49> Info: 任务下载服务初始化成功, IP=127.0.0.1, PORT=7612
<2010-05-31 14:02:49> Info: 初始化监听服务组 (共10个监听服务)...
<2010-05-31 14:02:49> Info: 监听服务启动成功, IP=127.0.0.1, PORT=6502
<2010-05-31 14:02:49> Info: 监听服务启动成功, IP=127.0.0.1, PORT=6503
<2010-05-31 14:02:49> Info: 监听服务启动成功, IP=127.0.0.1, PORT=6504
<2010-05-31 14:02:50> Info: 监听服务启动成功, IP=127.0.0.1, PORT=6505
<2010-05-31 14:02:50> Info: 监听服务启动成功, IP=127.0.0.1, PORT=6506
```

【Master 节点终止命令】

安全退出方式: 直接 Ctrl+C 或通过 KILL 命令退出 (只支持 Linux)

【配置文件简要说明】

Master 节点的配置文件名为 master.ini, 内容如下:

```
[basic]
# 当前 Master 节点的 IP 和端口
IP=127.0.0.1
Port=6501

# -----
# HTTP 服务方便管理员查看框架运行情况, 只支持在 Unix/Linux 下使用
# -----

[httpd]

# 非 0 时开启 HTTP 服务
Enabled=1

# HTTP 端口
```

```
Port=6580
# -----
# 以下为各个类型应用程序对应的任务生成程序,用户需要在这里为各个注册到当前 Master
# 节点的 Client 节点的应用程序类型配置其对应的任务生成程序, 格式为:
# [AppType]=[AppCmd], 如: "NewsGather=./NewsTaskCreate"表示 Client 节点应用程序类
# 型为"NewsGather"的任务生成程序为"./NewsStore"。其中:
#     ---- [AppType]表示 Client 节点的用户应用程序类型。
#     ---- [AppCmd]表示对应的任务生成程序名, 如"./NewsTaskCreate"。
# 当 Client 节点需要任务时, MARC 框架会自动调用相应类型的任务生成程序, 调用接口
# 为: [AppCmd] [TaskDirPath] [ClientID]
# 如"./NewsTaskCreate ./task/1_NewsGather_task_20100204095923/ 1", 其中:
#     ---- [TaskDirPath]用于存放任务生成程序的执行结果;
#     ---- [ClientID]是需要任务的 Client 节点 ID
# 注意几点:
# (1) 任务生成程序不需要自己创建[TaskDirPath]文件夹, MARC 框架会先行自动创建。
# (2) 任务生成程序执行成功后,须在[TaskDirPath]文件夹中生成".success"文件,否则 MARC
# 框架将不会将其下发给 Client 节点。
# (3) AppType 是大小写敏感的。
# -----
[appcmd]

#Client 节点的应用程序类型为"Test"的任务生成程序
Test=./TaskCreateTest.sh

#Client 节点的应用程序类型为"ForumGather"的任务生成程序
ForumGather=./ForumTaskCreate

#Client 节点的应用程序类型为"NewsGather"的任务生成程序
NewsGather=./NewsTaskCreate

# -----
# 以下部分用于进行应用程序版本升级。
# Master 节点将在[ClientUpdateDir]以及[ResultUpdateDir]下自动为当前活跃的所有 Client 节
# 点所属的 AppType 创建一个子目录,
# 管理员可在其子目录下放置升级包, 每个升级包对应一个文件夹, 以版本号命名, 最小版
# 本号是 1,
# 如"./update_client/NewsGather/2/"存放 AppType 为"NewsGather"的版本号为 2 的升级包文
# 件夹。
# 注意几点:
# (1) 管理员在确认升级包可用时, 须在升级包的文件夹中生成一个空的标志文件
# "update.ok", 否则框架将忽略该升级包。比如上述
#     版本号为 2 的升级包在管理员确认可用后, 应在"./update_client/NewsGather/2/"文件
# 夹下创建一个"update.ok"空文件。
# (2) 每个 AppType 的每个升级包版本不建议以 patch 补丁形式提供, 而是均为完整版本,
```

这样，在框架的长期运行过程中，中途新增的

- # Client 节点才能得到最完整的 App 版本（目前的实现机制是 Client 在向 Master 请求升级 App 时，只从 Master 上下载最新的升级包，
- # 然后替换 Client 节点上相应的文件或文件夹）。
- # (3) Master 会为每个升级包进行压缩，压缩的升级包文件存放在每个 AppType 的对应文件夹中，如"./update_client/NewsGather/2/"
- # 的升级包压缩文件为"NewsGather_2.myzip"，存放在"./update_client/NewsGather/"下。
- # (4) 管理员在创建新的升级包时，应确保其版本号（即升级包文件夹名称）大于之前的最大版本号（可通过升级包压缩文件名来判断）
- # (5) 如果同一时刻在某个 AppType 有多个版本号的升级包，则框架只保留版本号最大的那个升级包。
- # (6) 每个升级包中如果有可执行文件，须确保其具有可执行权限(Linux 下的"chmod +x"命令可修改成可执行)。
- # (7) 每个升级包压缩文件下载到 Client 节点或 Result 节点后，框架自动解压到 [UpdateTargetPath]下（参见 client.ini 和 result.ini）。

[update]

Client 节点应用程序升级包的存放路径
ClientUpdateDir=./update_client/

Result 节点应用程序升级包的存放路径
ResultUpdateDir=./update_result/

每隔多少秒扫描是否有应用程序升级包
VerWatchInterval=10

以下部分为 Master 节点高级配置，一般来说不用改动

[advanced]

最多的监听服务数（不小于 1）
MaxListenerCount=10

任务生成路径（存放执行任务生成程序后的结果）
TaskDir=./task/

任务压缩路径（存放压缩后的任务文件）
ZipTaskDir=./myzip_task/

任务生成机制

0: 只要 Client 节点没有待处理任务，就为其生成新任务
1: 只有等 Client 节点主动请求任务且当前没有待处理任务时才为其生成新任务
TaskCreateStrategy=0

已下发任务执行时长(从任务被下发到 Client 节点开始算起到当前时间)最大值(秒数)
超过该值的任务将被认为执行失败，从而从正处理任务对列中删除
MaxTaskRunTime=3600

多长时间保存一次 Client 节点应用程序状态(秒数)
MaxSaveStateTime=600

#任务下载时每个数据包的最大字节数（不小于 4096）
MaxPacketSize=4096

[AppCmd]中的各个任务生成程序的执行时限(秒)
AppRunTimeout=60

非 0 时自动删除已处理完的任务压缩文件
AutoDeleteTaskFile=1

设置失败任务的处理策略。一个任务失败可能由下列原因引起：
（1）Master 节点检测到任务下载超时（当参数 MaxTaskFetchTime 非 0 时），由 Master 节点自行判定该任务为失败任务；
（2）Master 节点检测到任务执行超时（当参数 MaxTaskRunTime 非 0 时），由 Master 节点自行判定该任务为失败任务；
（3）因某个 Client 节点正常退出或崩溃或网络异常等原因造成 Client 节点与 Master 的连接断开，由 Master 节点自行判定该 Client 节点当时正被执行的任务和后续待处理任务都是失败任务；
（4）Client 节点检测到任务没有被成功下载，由 Client 节点告知 Master 节点该任务为失败任务；
（5）Client 节点检测到任务执行程序运行超时，由 Client 节点告知 Master 节点该任务为失败任务；
（6）在 Client 节点上，因下列原因造成任务执行应用程序运行失败，由 Client 节点告知 Master 节点该任务失败：
---- 对下载完成的任务文件解压失败（检查 myzip 是否存在且有可执行权限）；
---- client.ini 中设置的任务执行命令[AppCmd]调用失败（请检查[AppCmd]是否能在命令行下单独运行成功）；
---- 任务执行程序运行结束，但当前目录下没发现.success 文件；
---- 任务执行完成，但结果文件夹[output]压缩失败（检查 myzip 是否存在且有可执行权限）；
参数的取值及含义：
0: 忽略该任务，不再处理（缺省）
1: 只能被同一个 Client 节点重新执行该任务（注：Master 节点基于 ClientID 判断是否为同一个 Client 节点）

```
# 2: 允许同一类型的其他 Client 节点（即 client.ini 中的 AppType 相同）重新执行该任务
TaskFailStrategy=2

# 任务失败时重试次数，超过则抛弃（该参数只有在 TaskFailStrategy 非 0 时才有意义）
TaskFailMaxRetry=3

# Master 节点终止时是否记录未处理和失败的任务信息
# 0 - 不记录，未处理和失败的任务被抛弃，Master 节点下次启动时不再重新载入
# 1 - 记录到临时文件，下次 Master 节点下次启动时重新载入
AutoSaveUnfinishedTask=1

# 每隔多长时间监控 Master 节点资源使用状态信息（秒）
SourceStatusInterval=300
```

5.3 Result 节点运行及配置文件简要说明

【Result 节点程序目录结构】

```
[gdding@localhost marc_result]$ ll
total 2136
-rwxr-xr-x 1 gdding gdding 1737513 May 31 13:57 marc
-rwxr-xr-x 1 gdding gdding 423243 May 31 13:57 myzip
-rw-rw-r-- 1 gdding gdding 4970 May 22 21:53 result.ini
-rwxrwxr-x 1 gdding gdding 111 Mar 24 09:40 ResultStoreTest.sh
```

其中 marc 文件即是 MARC 框架的主程序，result.ini 为 Result 节点的配置文件，myzip 为 MARC 框架中使用的内部压缩与解压工具，ResultStoreTest.sh 为 Linux 下用于测试的一个结果处理应用程序。

【Result 节点启动命令】

在 marc_result 目录下以 result 方式执行 marc 程序即可。如下图所示。

```
[gdding@localhost marc_result]$ ./marc result
<2010-05-31 14:16:44> Info: 准备启动Result节点...
<2010-05-31 14:16:44> Info: 注册到Master节点(127.0.0.1:6501)...
```

【Result 节点终止命令】

安全退出方式：直接 Ctrl+C 或通过 KILL 命令退出（只支持 Linux）

【配置文件简要说明】

Result 节点的配置文件名为 result.ini，内容如下：

```
[master]

# Master 节点的 IP 和端口
```

```
MasterIp=127.0.0.1
MasterPort=6501

# Master 备用节点的监听 IP 和端口
BakMasterIp=127.0.0.1
BakMasterPort=6501

[result_server]

# 当前 Result 节点的监听 IP 和端口
ListenIp=127.0.0.1
ListenPort=6601

# 非空时当前 Result 节点只接收指定类型（参阅 client.ini 中的 AppType 参数）的结果数据
# 不填写时可接收所有类型的应用程序的结果数据
AppType=

# -----
# 各种类型应用程序的结果处理程序，格式为: [AppType]=[AppCmd],
# 其中[AppType]为用户应用程序的类型（对应 client.ini 中的 AppType 参数），
# [AppCmd]为用户针对该类型应用程序而编写的结果处理程序。
# 注意几点：
# (1) 若上面的 ResultType 参数非空，则这里应列出所有应用程序类型的处理程序；否则可
# 只列出 ResultType 对应的处理程序。
# (2) Result 节点接收到结果数据后，结果处理程序会被自动调用，调用接口：[AppCmd]
# [ResultPath] [ClientID]
# 如"./NewsStore ./data/1_NewsGather_result_20100204095923/ 1"，其中：
# ---- [ResultPath]为待处理结果的存放路径，其存放的内容对应于 Client 节点的
# "OutputDir"文件夹内容。
# ---- [ClientID]为生成该结果的 Client 节点 ID。
# (3) 结果处理程序执行成功后，须在[ResultPath]目录下生成.success 标志文件告知 MARC
# 框架其执行成功。
# (4) 结果处理程序执行后（不管是否成功），MARC 框架会自动删除结果数据文件夹。
# (5) 若结果处理程序执行失败，则 MARC 框架会将失败的结果文件重新放入待处理对列等
# 待以后再处理。
# -----
[appcmd]

#应用程序类型为"Test"的结果处理程序
Test=./ResultStoreTest.sh

#应用程序类型为"NewsGather"的结果处理程序
NewsGather=./NewsStore
```



```
#应用程序类型为"ForumGather"的结果处理程序
ForumGather=./ForumStore

# -----
#以下部分用于进行应用程序版本更新的相关参数设置
# -----

[update]

# 升级目标路径（即升级包中的文件存放在何处）
UpdateTargetPath=./

# 每隔多少秒向 Master 请求是否需要升级
UpdateInterval=60

# 下载的升级包压缩文件的临时存放路径
ZipUpdateDir=./myzip_update_download/
# -----
# 以下部分为 Result 节点的高级配置，一般来说不用改动
# -----

[advanced]

# 存放从 Client 节点接收并解压后的结果数据的临时文件夹（已解压）
DataDir=./data/

# 从 Client 节点接收的结果压缩文件的缓存文件夹
ZipResultDir=./myzip_result/

# Result 节点心跳周期（每隔多少秒发送一次心跳信息以及状态信息）
HeartbeatInterval=3

#上传时每个数据包的最大字节数（不小于 4096）
MaxPacketSize=4096

# 各个结果处理程序的执行时限(秒)
AppRunTimeout=60

#非 0 时，自动删除已处理完的结果文件
AutoDeleteResultFile=1

# 非 0 时在 Result 节点终止时自动保存未处理的结果文件以便下次启动时重新载入
AutoSaveUnfinishedResultFile=1

# 每隔多长时间监控 Result 节点资源使用状态信息（秒）
```

SourceStatusInterval=300

5.4 Client 节点运行及配置文件简要说明

【Client 节点程序目录结构】

```
[gdding@localhost marc_client]$ ll
total 2144
-rw-rw-r-- 1 gdding gdding 6016 May 22 21:53 client.ini
-rw-rw-r-- 1 gdding gdding 196 Mar 26 10:03 killprolist.ini
-rwxrwxr-x 1 gdding gdding 153 Mar 25 13:04 killpro.sh
-rwxr-xr-x 1 gdding gdding 1737513 May 31 13:57 marc
-rwxr-xr-x 1 gdding gdding 423243 May 31 13:57 myzip
-rwxrwxr-x 1 gdding gdding 138 Mar 19 09:46 Test.sh
```

其中 marc 文件即是 MARC 框架的主程序，client.ini 为 Client 节点的配置文件，myzip 为 MARC 框架中使用的内部压缩与解压工具，Test.sh 为 Linux 下用于测试的一个任务执行应用程序，killpro.sh 以及 killprolist.ini 为 MARC 框架在特殊情况用于杀死任务执行应用程序的脚本和相应的配置文件。

【Client 节点启动命令】

在 marc_client 目录下以 client 方式执行 marc 程序即可。如下图所示。

```
[gdding@localhost marc_client]$ ./marc client
<2010-05-31 14:25:02> Info: 准备启动Client节点...
<2010-05-31 14:25:02> Info: 注册到Master节点(127.0.0.1:6501)...
```

【Client 节点终止命令】

安全退出方式：直接 Ctrl+C 或通过 KILL 命令退出（只支持 Linux）

【配置文件简要说明】

Client 节点的配置文件名为 client.ini，内容如下：

```
[master]

# Master 节点的监听地址
MasterIp=127.0.0.1
MasterPort=6501

# Master 备用节点的监听地址
BakMasterIp=127.0.0.1
BakMasterPort=6501

[client]
```

Client 节点编号,数字类型,在整个 MARC 框架中唯一。为 0 时由 Master 自动分配 ID。
由 Master 来分配 ID 可以节省人工配置工作量以及容易出错的问题,并确保所有 Client 都能得到一个唯一的 ID。
但要注意,同一个 Client 在每次启动时分配得到的 ID 很可能不相同,有些应用需要为各个 Client 节点设置固定的 ID,在这种情况下,建议不要由 Master 来分配 ID。
Master 在自动分配 ID 时,从 65536 开始,若用户需自己设置固定的 ID,建议设置为 65536 以下的值。

ID=0

被调用的用于执行任务的用户应用程序
调用方式: [AppCmd] [InputDir] [OutputDir] [ID], 如"./Test.exe ./input/ ./output/ 2"
其中:
---- [AppCmd]可以是可执行程序或批处理程序或 shell 脚本 (Linux 下应注意加上路径如"./Test.sh"或"./Test")
---- [InputDir]对应于下面 InputDir 参数给定的文件夹路径,用于存放用户应用程序的输入,对应于 MARC 框架的 Master 节点的任务生成程序来创建的任务生成结果文件夹。
---- [OutputDir]对应于下面的 OutputDir 参数给定的文件夹路径,用于存放用户应用程序的执行结果。用户应用程序执行完成后, MARC 框架将对[OutputDir]中的文件进行压缩,回传给 Result 节点。
注意:
(1) 用户应用程序必须能在有限时间内执行完成,不能是长期运行的“循环型”程序。
(2) 用户应用程序执行成功后,须在 MARC 框架的当前目录下生成空的标志文件#
".success", 以向 MARC 框架表明程序执行成功。
(3) 若 MARC 框架找不到.success 文件则认为程序执行失败, Master 节点将认为该任务执行失败 (失败的任务后续会重新下发和执行)
(4) 若 AppCmd 为空或文件不存在,则不会向 Master 节点请求任务,此时 Client 节点从功能上看只用来发送节点状态信息

AppCmd=./Test.sh

被调用的用户应用程序的类型,须是由字母或数字组成的字符串 (不能含有下划线_),不能是空串。
AppType 在整个 MARC 框架中起着十分重要的作用,它为具有相同的任务生成、任务执行、结果处理的用户应用程序起着关联的作用。在整个 MARC 框架中,支持多种不同类型的用户应用程序,也就是说,不同类型的用户应用程序可以部署于同一个 MARC 框架中,从而完成多种不同类型的任务。

AppType=Test

以下部分用于进行应用程序版本更新的相关参数设置。

```
# Client 节点在空闲时（即未执行任务时），每隔一段时间（由参数[UpdateInterval]指定）向
# Master 请求是否需要升级，若需要升级则从 Master 节点上下载版本号最大的升级包压缩文
# 件，下载完成后解压到参数[UpdateTargetPath]指定的路径下。
# 注意几点：
# (1) 升级只针对应用程序（及其相关文件），不能对框架本身进行升级。
# (2) 是否需要升级的依据：Master 节点上有一个升级包，其版本号大于 Client 节点的应用
#     程序当前版本号。
# (3) 升级完成后，MARC 框架会自动把升级后的版本号保存在本地的[AppType].marc.ver
#     文件中。
# -----
```

[update]

```
# 升级目标路径（即升级包中的文件存放在何处）
```

```
UpdateTargetPath=./
```

```
# 每隔多少秒向 Master 请求是否需要升级
```

```
UpdateInterval=600
```

```
# 下载的升级包压缩文件的临时存放路径
```

```
ZipUpdateDir=./myzip_update_download/
```

```
# -----
```

```
#以下部分为 Client 节点的高级配置，一般来说不用改动
```

```
# -----
```

[advanced]

```
# 用户应用程序的输入文件夹路径
```

```
InputDir=./input/
```

```
# 用户应用程序的输出文件夹路径
```

```
OutputDir=./output/
```

```
# 从 Master 节点接收的任务压缩文件的临时存放路径
```

```
# 系统自动将任务压缩文件解压到[InputDir]
```

```
ZipTaskDir=./myzip_task_download/
```

```
# 对用户应用程序执行结果（即[OutputDir]）压缩后的存放路径
```

```
ZipResultDir=./myzip_result_upload/
```

```
# Client 节点心跳周期（每隔多少秒发送一次心跳信息以及状态信息）
```

```
HeartbeatInterval=10
```

```
# Client 节点应用程序状态发送时间间隔（秒）
StateInterval=60

# 用户应用程序的最大运行时间（单位秒），超过该值时将被 MARC 框架强行杀死
# 若设置为 0 则不限定程序运行时间
AppRunTimeout=3600

# 0：同步上传，当任务完成后，只有等结果上传完毕后才请求下一次任务
# 1：异步上传，当任务完成后，立即请求下一次任务，结果上传将异步进行
AsynUpload=1

# 异步上传时，等待上传的文件个数最大值（若超过该值，Client 节点将不再向 Master 节点
# 获取任务）
MaxWaitingUploadFiles=5

#非 0 时，自动删除已上传完的结果文件
AutoDeleteResultFile=1

#非 0 时，自动删除已处理完的任务文件
AutoDeleteTaskFile=1

#非 0 时，记录首次上传结果时向 Master 请求获得当时负载最小的 Result 节点地址
#此后上传时不再向 Master 请求获得 Result 节点地址，除非某次上传失败。
#为 0 时，每次上传结果都将向 Master 节点请求获得当时负载最小的 Result 节点地址
RememberResultAddr=0

# Client 向 Master 请求任务失败时隔多少秒再次请求
# 请求失败的原因一般是因为请求时 Master 节点上没有准备好任务
TaskReqTimeInterval=5

# 非 0 时在 Client 节点终止时自动保存未上传的结果文件以便下次启动时重新载入
AutoSaveUploadFile=1

# 每隔多长时间监控 Client 节点资源使用状态信息（秒）
SourceStatusInterval=300
```

六、前台管理界面

MARC 框架实现了一个内嵌的 HTTP Server，使得框架的用户或系统运维人员能方便地通过浏览器实时地查看框架中各个节点的运行状态、资源使用情况、任务执行状况、日志信息等。以下简要对前台界面进行介绍。

6.1 首页

内嵌的 HTTP Server 集成在 Master 节点上，当 Master 节点启动后，就能通过浏览器查看前台界面（须通过设置 master.ini 启用 HTTP Server，此项功能目前不支持 WINDOWS）。

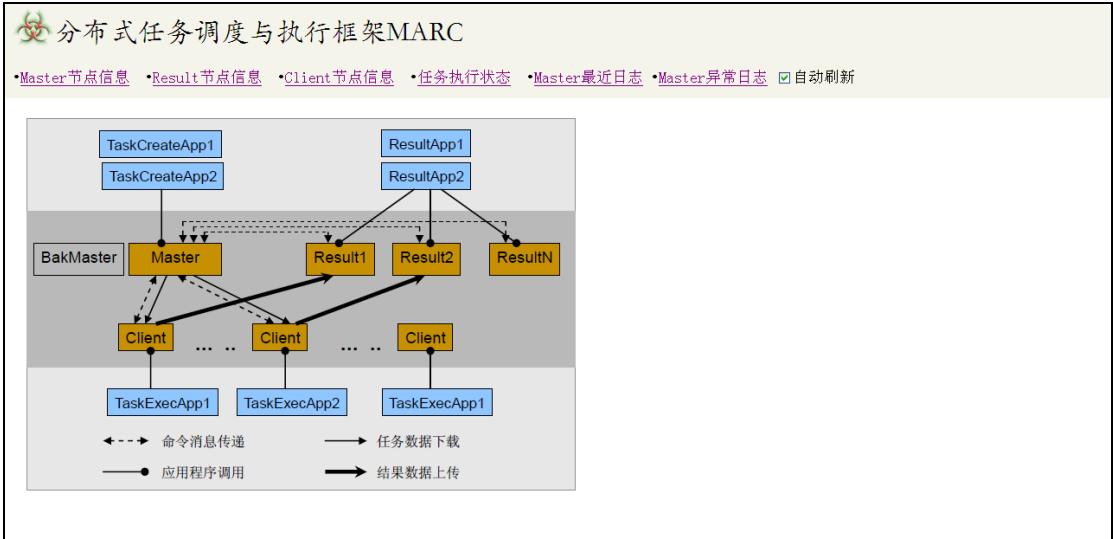


图 6.1 MARC 框架前台界面首页

6.2 Master 节点信息

图 6.2 Master 节点信息页面展示了 Master 节点的相关信息。页面顶部是标题栏，包含“分布式任务调度与执行框架MARC”和一系列链接。下方是信息列表：

- 主机地址：**10.61.1.245:6501
- 部署位置：**/home/gdding/marc/m/
- 启动时间：**2011-07-22 12:42:04 【已运行 0 天 0 小时 46 分钟】
- 节点数目：**Result节点 9个，Client节点 125个
- 创建任务数：**975个，平均创建时间：0秒
- 最近创建任务：**ID=1075，任务文件为./myzip_task/10038_Test_task_20110722132842.myzip
- 任务执行情况：**分发任务数：1071个，完成任务数：950个，失败任务数：0个
- 资源使用情况：**CPU 空闲率：100%，磁盘剩余率 85%，内存空闲率 17%，网卡速率：47280 bps

图 6.2 Master 节点信息页面

6.3 Result 节点信息




图 6.3 Result 节点信息页面

6.4 Client 节点信息



图 6.4 Client 节点信息页面

6.5 任务执行状态

 分布式任务调度与执行框架MARC

•Master节点信息 •Result节点信息 •Client节点信息 •任务执行状态 •Master最近日志 •Master异常日志 ☒ 自动刷新

任务状态列表

| 任务ID 类型 分配给 | 任务生成时间 | 任务请求时间 | 下载完成时间 | 执行完成时间 | 任务状态 |
|----------------------------|-------------------------|---------------------|-------------------------|------------|---------|
| 110508 Test,ClientID=10001 | 2011-07-25 17:01:15(0秒) | ----- | ----- (0秒) | ----- (0秒) | 等待下发... |
| 110507 Test,ClientID=10016 | 2011-07-25 17:01:14(1秒) | 2011-07-25 17:01:15 | ----- (0秒) | ----- (0秒) | 正在下发... |
| 110506 Test,ClientID=10085 | 2011-07-25 17:01:12(0秒) | 2011-07-25 17:01:13 | 2011-07-25 17:01:16(3秒) | ----- (0秒) | 正在执行... |
| 110505 Test,ClientID=10036 | 2011-07-25 17:01:12(0秒) | 2011-07-25 17:01:13 | 2011-07-25 17:01:16(3秒) | ----- (0秒) | 正在执行... |
| 110504 Test,ClientID=10082 | 2011-07-25 17:01:12(1秒) | 2011-07-25 17:01:12 | 2011-07-25 17:01:15(3秒) | ----- (0秒) | 正在执行... |
| 110503 Test,ClientID=10086 | 2011-07-25 17:01:05(0秒) | 2011-07-25 17:01:11 | 2011-07-25 17:01:14(3秒) | ----- (0秒) | 正在执行... |
| 110502 Test,ClientID=10079 | 2011-07-25 17:01:05(1秒) | 2011-07-25 17:01:05 | 2011-07-25 17:01:08(3秒) | ----- (0秒) | 正在执行... |
| 110501 Test,ClientID=10099 | 2011-07-25 17:01:01(0秒) | 2011-07-25 17:01:01 | 2011-07-25 17:01:04(3秒) | ----- (0秒) | 正在执行... |
| 110500 Test,ClientID=10080 | 2011-07-25 17:01:01(1秒) | 2011-07-25 17:01:07 | 2011-07-25 17:01:10(3秒) | ----- (0秒) | 正在执行... |
| 110499 Test,ClientID=10087 | 2011-07-25 17:00:57(0秒) | 2011-07-25 17:00:58 | 2011-07-25 17:01:01(3秒) | ----- (0秒) | 正在执行... |
| 110498 Test,ClientID=10074 | 2011-07-25 17:00:55(0秒) | 2011-07-25 17:00:56 | 2011-07-25 17:00:59(3秒) | ----- (0秒) | 正在执行... |
| 110497 Test,ClientID=10071 | 2011-07-25 17:00:47(0秒) | 2011-07-25 17:00:48 | 2011-07-25 17:00:51(3秒) | ----- (0秒) | 正在执行... |
| 110496 Test,ClientID=10083 | 2011-07-25 17:00:47(0秒) | 2011-07-25 17:00:53 | 2011-07-25 17:00:56(3秒) | ----- (0秒) | 正在执行... |
| 110495 Test,ClientID=10068 | 2011-07-25 17:00:47(1秒) | 2011-07-25 17:00:51 | 2011-07-25 17:00:54(3秒) | ----- (0秒) | 正在执行... |
| 110494 Test,ClientID=10065 | 2011-07-25 17:00:46(0秒) | 2011-07-25 17:00:46 | 2011-07-25 17:00:49(3秒) | ----- (0秒) | 正在执行... |
| 110493 Test,ClientID=10056 | 2011-07-25 17:00:46(1秒) | 2011-07-25 17:00:46 | 2011-07-25 17:00:49(3秒) | ----- (0秒) | 正在执行... |

图 6.5 任务执行状态页面

6.6 Master 日志信息

 分布式任务调度与执行框架MARC

•Master节点信息 •Result节点信息 •Client节点信息 •任务执行状态 •Master最近日志 •Master异常日志

• <2011-07-15 14:40:57> Info: 准备为Client节点 (ID=2001, AppType=Test) 执行任务生成程序"/TaskCreateTest.sh"...

• <2011-07-15 14:40:58> Info: Client节点 (ID=2001, AppType=Test, IP=10.61.1.249) 请求任务...

• <2011-07-15 14:40:58> Info: Client节点 (ID=2001, AppType=Test, IP=10.61.1.249) 当前没有任务要执行

• <2011-07-15 14:41:03> Info: 为Client节点 (ID=2001, AppType=Test) 创建任务成功, 生成的任务为./myzip_task/2001_Test_task_20110715144057.myzip

• <2011-07-15 14:41:04> Info: Client节点 (ID=10001, AppType=Test, IP=10.61.1.249) 告知其任务执行程序运行正常结束

• <2011-07-15 14:41:04> Info: Client节点 (ID=2001, AppType=Test, IP=10.61.1.249) 请求任务...

• <2011-07-15 14:41:04> Info: 为该节点取得待处理任务 (TaskID=269, TaskFile=./myzip_task/2001_Test_task_20110715144057.myzip)

• <2011-07-15 14:41:04> Info: Client节点 (ID=10001, AppType=Test, IP=10.61.1.249) 请求上传结果...

• <2011-07-15 14:41:04> Info: 结果需上传到Result节点 (127.0.0.1:6601|./myzip_result/)

• <2011-07-15 14:41:05> Info: 准备为Client节点 (ID=10001, AppType=Test) 执行任务生成程序"/TaskCreateTest.sh"...

• <2011-07-15 14:41:05> Info: Client节点 (ID=10001, AppType=Test, IP=10.61.1.249) 请求任务...

• <2011-07-15 14:41:05> Info: Client节点 (ID=10001, AppType=Test, IP=10.61.1.249) 当前没有任务要执行

• <2011-07-15 14:41:07> Info: Client节点 (ID=2001, AppType=Test, IP=10.61.1.249) 告知其任务下载成功 (TaskID=269)

• <2011-07-15 14:41:10> Info: 为Client节点 (ID=10001, AppType=Test) 创建任务成功, 生成的任务为./myzip_task/10001_Test_task_20110715144105.myzip

• <2011-07-15 14:41:11> Info: Client节点 (ID=10001, AppType=Test, IP=10.61.1.249) 请求任务...

• <2011-07-15 14:41:11> Info: 为该节点取得待处理任务 (TaskID=270, TaskFile=./myzip_task/10001_Test_task_20110715144105.myzip)

• <2011-07-15 14:41:14> Info: Client节点 (ID=10001, AppType=Test, IP=10.61.1.249) 告知其任务下载成功 (TaskID=270)

• <2011-07-15 14:41:26> Info: Client节点 (ID=10001, AppType=Test, IP=10.61.1.249) 告知其任务执行程序运行正常结束

• <2011-07-15 14:41:27> Info: Client节点 (ID=10001, AppType=Test, IP=10.61.1.249) 请求上传结果...

• <2011-07-15 14:41:27> Info: 结果需上传到Result节点 (127.0.0.1:6601|./myzip_result/)

• <2011-07-15 14:41:27> Info: 准备为Client节点 (ID=10001, AppType=Test) 执行任务生成程序"/TaskCreateTest.sh"...

• <2011-07-15 14:41:28> Info: Client节点 (ID=10001, AppType=Test, IP=10.61.1.249) 请求任务...

• <2011-07-15 14:41:28> Info: Client节点 (ID=10001, AppType=Test, IP=10.61.1.249) 当前没有任务要执行

• <2011-07-15 14:41:29> Info: Client节点 (ID=2001, AppType=Test, IP=10.61.1.249) 告知其任务执行程序运行正常结束

• <2011-07-15 14:41:30> Info: Client节点 (ID=2001, AppType=Test, IP=10.61.1.249) 请求上传结果...

图 6.6 Master 日志信息页面

6.7 Master 异常日志信息



图 6.7 Master 异常日志信息页面

附录 1 系统开发简介

MARC 的前身为在某项目中开发出来的 dfs-3.0。由于开发时间太紧以及缺乏后续的完善支持，dfs-3.0 在代码质量、系统性能、容错性等方面存在一些不如人意的地方。MARC 系统是在 dfs-3.0 的基础上通过不断的代码重构、功能完善、性能优化、接口简化、容错改进、日志完善以及不断的测试等而进化过来的，可以认为是 4.0 版本的 dfs，但在接口使用上与 dfs-3.0 有一些区别，此处不对 dfs-3.0 进行介绍，仅列出一些在 dfs-3.0 基础上的重要的改进之处。

■ 代码重构

- ✓ MARC 是 dfs-3.0 进化过来的，借鉴了 dfs-3.0 中的大部分的设计理念和流程以及代码，经过 N 次的反复重构和代码修改，如今几乎看不到在代码中与 dfs-3.0 有相同的地方，基本上是重写的。
- ✓ 与 dfs-3.0 相比，MARC 系统在代码质量、代码逻辑性、代码可读性、模块化设计、注释、系统性能、容错性、可扩展性、接口易用性、可维护性等方面都有了较大的提升。

■ 功能完善及性能优化

- ✓ 增加了异步上传功能：在 client 节点上增加任务执行结果的异步上传功能，达到任务下载及执行、结果回传这两个执行序列的并行化的效果，这样不必等待一次任务执行结果上传完成后才去请求下一次任务，增强了 MARC 系统整体的实时性。
- ✓ 通信消息的协议完善与优化：对节点之间的命令消息的格式、消息种类、消息交互过程进行了完善和优化。
- ✓ 任务下载及结果传输的效率优化：之前的 dfs-3.0 版本在传输数据时（如任务下载和结果回传）由于采用了简单的消息控制协议，导致网络带宽的利用率浪费了一倍，通过完善数据传输协议并增加可配置性，使得整体系统的带宽利用率得到了很大提升，并可以根据不同网络的带宽情况在配置文件中来调整。
- ✓ 解决网络字节序问题：dfs-3.0 版本中没有解决不同硬件平台的网络字节序问题，在当前版本 4.0 的 MARC 中，有效地解决了这个问题（感谢朱斌的努力）。
- ✓ Myzip 模块的跨平台实现的代码统一和压缩、解压策略的优化（感谢朱斌的努力）。
- ✓ Result 节点使用异步方式来对接收到的结果数据进行处理：启动一个结

果处理线程对待处理的每个结果压缩文件进行处理，若处理失败则重新加入对列；若主动退出则将未处理完毕的文件名写入临时文档等待下次启动后再进行处理。

■ 容错性及鲁棒性的完善

- ✓ 应用程序的调用采用 `fork` 子进程方式（可执行 `shell` 脚本，Windows 平台下则使用 `ShellExecuteEx` 来创建子进程，若使用 `CreateProcess` 似乎无法执行批处理命令），便于监测和控制程序运行过程。
- ✓ 增加基于任务 ID 的任务执行状况的记录功能
- ✓ 完善各种错误及异常处理
- ✓ 系统启动时应将上次未上传或上传失败的结果文件重新上传
- ✓ 记录每个连接的建立时间，对于短连接，监控其活动时间，若超时则关闭该连接。
- ✓ Master 节点通过记录 Client 节点、Result 节点的心跳连接来监测节点是否出现故障。
- ✓ 增加自动将失败任务转给其他 Client 节点处理的功能，基本机制是：(1) 使用一个任务对列来保存所有失败或超时的任务（含各种应用类型的所有节点的失败任务）；(2) 判断某个节点是否有待处理任务时，除了需要根据该节点的待处理任务对列来判断外，还需要从失败任务对列中查找（应用类型应相同）；(3) 当某个 Client 节点发生故障时，自动将该节点的所有未处理完成的任务加入失败任务对列；(4) 当某个 client 节点处理某个任务失败时，自动将该任务加入失败任务对列；(5) 当某个 client 节点需要获取任务时，优先从失败任务对列中获取（应用类型相同），然后将获取到的任务加入该节点自己的待处理任务对列。

■ 易用性及易维护性的完善

- ✓ 在配置文件中剔除了无关的参数，简化系统的配置，修改一些参数的命名，提高配置文件的可理解性；
- ✓ 所有的应用程序是由 MARC 自动调用的，不需要人工启动或干预。
- ✓ 允许在 client 端不配置节点 ID 也让整个系统运行正常。当 Client 节点很多时，节点 ID 配置工作量较大，且容易出错。为兼顾不同应用的需求（有些应用需要尽可能保证任务与节点之间的分配关系是静态，如定向采集系统，需要保证同一个 Client 节点在前后两次启动时得到的 ID 要相同，否则就不能用基于 Client 节点 ID 的方式来分配任务；而有些应用则没有这种需求），按照如下策略完善 MARC：Client 节点的 ID 配置功能保留，当用户将 ID 设置为 0 时，表示由 MARC 自动为 Client 节点分配 ID，在

这种情况下，在 Client 向 Master 注册阶段，由 Master 分配一个 ID 给 Client，这样可以确保所有 Client 都能得到一个唯一的 ID，且不需要人工配置。

- ✓ 系统运行日志的完善：对日志信息的表述等进行了完善，增强日志的可读性，便于系统管理员进行分析。
- ✓ 通过内嵌的 HTTP Server，为系统管理员或运维人员提供一个基于浏览器就可以查看整个 MARC 框架中各个节点运行状况的界面。如：当前有多少节点在运行，各个节点的基本信息：IP、功能、部署的程序等；资源使用情况；各个节点的资源使用情况；Client 节点的任务执行情况；Result 节点的结果处理情况；等等。

附录 2. 2011 年 7 月升级与完善列表

1. 改善框架的容错性

- (1) 完善内部通信协议；
- (2) 增加运行时对 SIGTERM/SIGINT 等信号的处理（只针对 Linux），防止用户通过 CTRL+C 或 KILL 指令强行退出，有利于提高数据安全性。
- (3) 增加 Result 节点向 Master 注册时的魔数校验功能；
- (4) 解决极端情况下的几个 BUG；
- (5) Master 节点配置文件增加 MaxTaskFetchTime 参数用于设置任务下发最大时长，缺省为 300 秒；
- (6) 当检测到任务下载用时超过最大时限或任务执行用时超过最大时限，Master 节点即认为该任务执行失败；
- (7) Result 节点退出时自动保存未处理完的结果文件，下次启动时重新载入；
- (8) Master 节点退出时自动保存未处理的任务信息，下次启动时重新载入；
- (9) 完善对失败任务的处理策略（配置文件 master.ini 可进行设置）；
- (10) 在多个 Result 节点的情况下，优化 Result 节点选取算法；
- (11) Master 在进行节点管理时，对 Result 节点和 Client 节点的删除并非真正删除而是设置标志位（这样前台能方便看到哪个节点已失效）。
- (12) Result 节点增加 ID，基于 ID 进行 Result 节点管理和与 Master 节点的通讯；

2. 完善各个节点执行状态、任务执行状况的记录和管理

- (1) 每个任务增加 CreatedTime 和 RequestTime 两个属性，分别用于记录任务的创建时间和被 Client 节点请求的时间；
- (2) Client 节点请求任务时，只请求 RequestTime 为 0 的任务，防止同一个

任务被重复请求；

(3) 完善 Result 节点和 Client 节点的运行状态记录，心跳时同时将节点状态发送给 Master，便于 Master 掌握各个节点的运行状态。

(4) Result 节点在启动时告知其部署路径给 Master 节点；

(5) 增加对 Master 节点各种执行状态的记录：已创建任务数、平均创建任务时间、已下发任务数、已完成任务数、失败任务数，等等；

3. 增强代码可读性

(1) 通过一定程度的代码重构，使得内部逻辑更加清晰；

4. 易用性和可维护性的大大增强，增加基于浏览器的前台展示功能和各个节点上的应用程序版本自动升级功能

(1) MARC 内嵌一个 HTTP Server，能方便得通过浏览器查看框架中各个节点的运行状态、任务执行状况、日志信息等，可以大大减少运维工作量。

(2) 增加所有节点的资源使用情况的监控和前台展示功能，包括 CPU、内存、磁盘、网络等，在前台能方便查看。

(3) 新版本的 MARC 框架在接口、部署上与之前版本完全相同，新增的 HTTP SERVER 不依赖于外部 Web Server。

(4) 增加运行时对 SIGTERM/SIGINT 等信号的处理（只针对 Linux），放置用户通过 CTRL+C 或 KILL 指令强行退出。

(5) Master 节点增加参数允许用户设置是否自动删除已完成的任务文件。

(6) 增加 Client 节点和 Result 节点上的应用程序自动升级功能。

附录 3 待开发的功能及完善列表

1. 完善 sftp_client 模块，目前上传或下载一个文件都用一个新连接，应增加在一个连接中可下载或上传多个文件的功能。
2. 采用高性能的网络通讯模型（如 Linux 下的 epoll、Windows 上的完成端口模型等）替换 select 模型以支持高并发。
3. 完善 Client 节点的应用程序状态发送以及 Master 节点状态接收功能；