分布式计算平台 BGN 介绍

目录

	联系人	电话	Email	时间	版本	备注
编写	周超勇	13910123864	bgnvendor@gmail.com	2013.08.17	v0.1	初稿
审核						

1、 平台定位

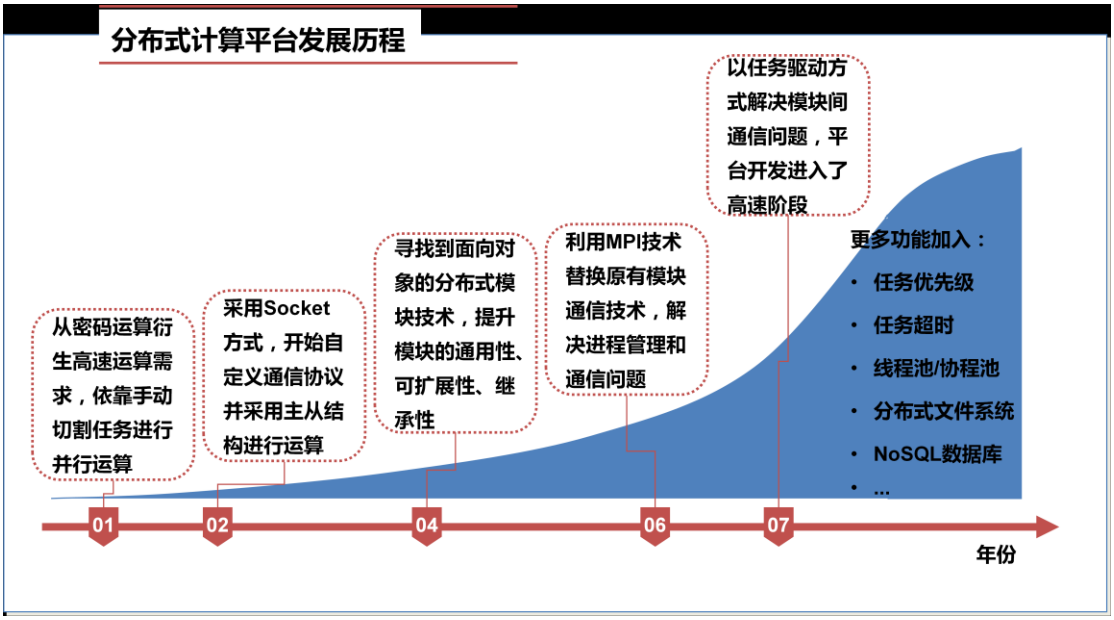
分布式计算平台（以下简称平台）是一款开源的通用并行计算和分布式计算平台软件，以降低并行和分布式应用的开发和部署技术门槛，具有资源池、无中心、大通道等特性。用户可通过增加新的模块不断加强平台能力，拓展平台的适用领域。

平台采用分层设计，将应用的整个运行环境分为应用层、平台层、环境层，分别对应云计算中的 IaaS、PaaS、SaaS，平台位于平台层。不同的应用或功能以模块的方式嵌入或载入平台，并通过平台实现彼此交互，外部应用以网络通信的形式接入平台组成的集群；平台通过适配层来适配底层不同的软硬件环境。

2、 平台特点

1、 纯 C 实现，面向对象思想，分层设计，约定通信协议
2、 目前支持 Linux/Unix 操作系统，支持异构（但不支持 32 位系统与 64 位系统之间的异构）
3、 支持异步无阻塞通信机制
4、 支持任务驱动，任务发射后不管，用户无须关心底层细节
5、 支持任务分裂、任务同步、任务自动指派、任务负载均衡、任务超时
6、 支持插接无限可扩展的应用或功能模块，并自动成为分布式模块，提供并行和分布式计算服务，支持模块继承
7、 串行程序转换为并行程序简单易行（但用户必须承认并牢记：具体问题的并行分解工作由人工完成，而非软件或平台）
8、 支持高并发、高扩展性、无中心部署、灵活组网

3、 平台发展历程



以上为简史。特别指出的是，2004 年是最重要的时间点，这一年分布式模块技术雏形诞生。

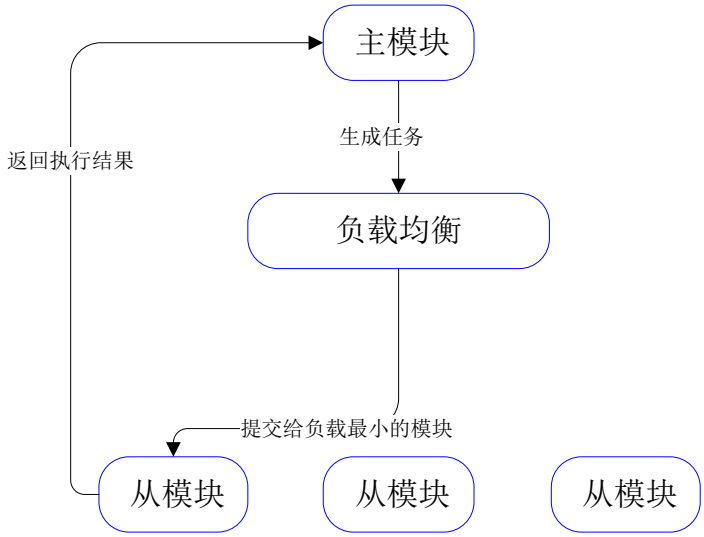
4、 平台编程模型

4.1 主从编程模型

比如分布式文件系统的 NameNode、DataNode 和 Client（对于分布式文件系统的三层设计方案，还有 NameSpace）。Client 对文件操作时，Client 是主，NameNode、DataNode 是从；多个 NameNode 流式写元数据信息时，前一个 NameNode 是主，接下来的一个 NameNode 是从；DataNode 流式写数据信息时，前一个 DataNode 是主，接下来的一个 DataNode 是从。

比如 MapReduce 也是典型的主从模型。

主从编程模型如下所示



图例说明：

- ✓ 主模块生成任务请求，提交任务请求以及收集任务响应
- ✓ 主模块控制并行时间线，决定下一轮任务的提交时间点
- ✓ 负载均衡维护从模块的负载装况，选择负载最小的模块提交任务请求
- ✓ 从模块执行任务请求，反馈任务响应

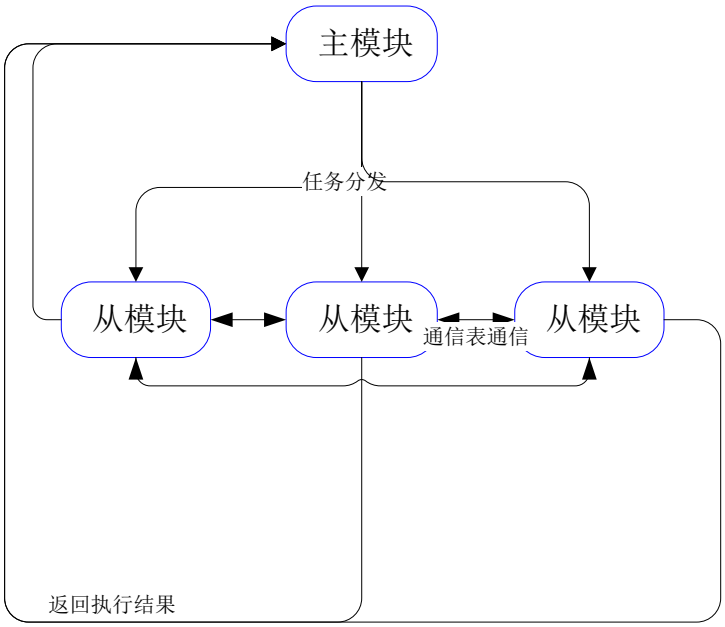
【注】

主从模型中的负载均衡是可选的，即主模块可以直接指定某个从模块执行任务请求。

4.2 通信表编程模型

比如某些矩阵乘算法中，中途需要交换数据，而且交换数据的通信双方或多方都是确定的，此时适合点到点的通信表编程模型。

通信表编程模型如下



图例说明：

- ✓ 主模块生成任务请求，提交任务请求以及收集任务响应
- ✓ 主模块控制并行时间线，决定下一轮任务的提交时间点
- ✓ 从模块执行任务请求，反馈任务响应
- ✓ 从模块根据通信表自主通信，交换任务执行中间结果

【注】

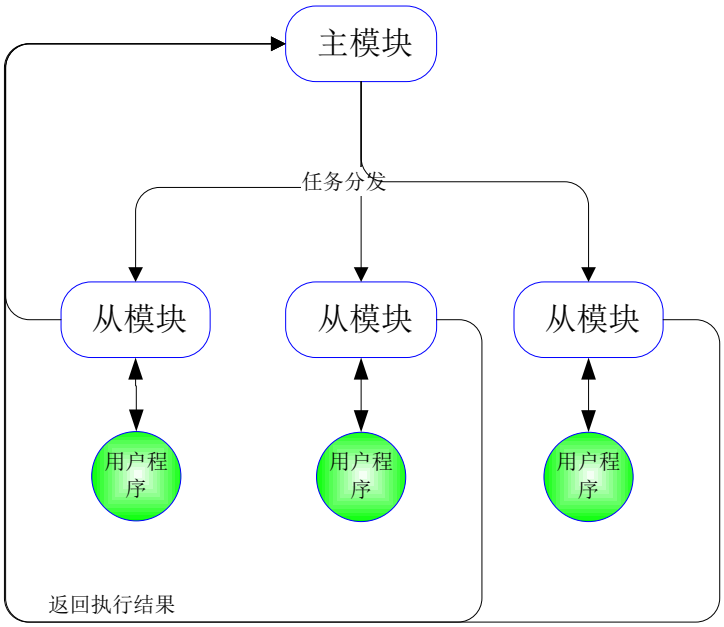
1. 通信表模型中的主模块是可选的，部分应用不需要多轮计算时可去掉主模块。
2. 通信表模型中从模块之间的通信隐含着主从通信模型

通信表模型在多种应用场合下可以获得良好的加速比。

5.3 整体调度编程模型

比如用户已有一个可执行文件能处理一段数据，现在有大量的数据可切分成更小的段交由可执行文件执行，此时可以通过整体调度编程模型，启用多个模块，每个模块加载一次该可执行文件，并发或分布的处理数据。
最简单的应用场景如，在每个物理节点上放上脚本，平台执行脚本，然后把各物理节点上的输出结果反馈回来。

整体调度编程模型如下



图例说明：

- ✓ 主模块生成任务请求，提交任务请求以及收集任务响应
- ✓ 主模块控制并行时间线，决定下一轮任务的提交时间点
- ✓ 从模块执行任务请求，反馈任务响应
- ✓ 从模块任务执行通过调用用户程序（库、模块或可执行文件）完成

【注】

1. 整体调度模型将用户程序视为一个功能模块
2. 整体调度模型中每个从模块执行同样的用户程序
3. 整体调度模型中的从模块之间不存在通信

一般而言，负载均衡在整体调度模型中是多余的或者通过其它方式进行

5、 平台概念

1、 模块定义：一组数据以及对这组数据进行操作的集合。

2、 主从模块定义：发起任务模块称为主模块，处理任务的模块成为从模块。事实上，用户应用也可以发起任务，也是主，处理任务的一定是从模块。

3、 一个远端模块的标识由任务通信子标识(tcid)、【MPI 通信子标识(comm)】、进程号(rank)、模块标识（modi）组成。其中，【MPI 通信子（comm)】是为了兼容 MPI 通信，当前平台已弃用。

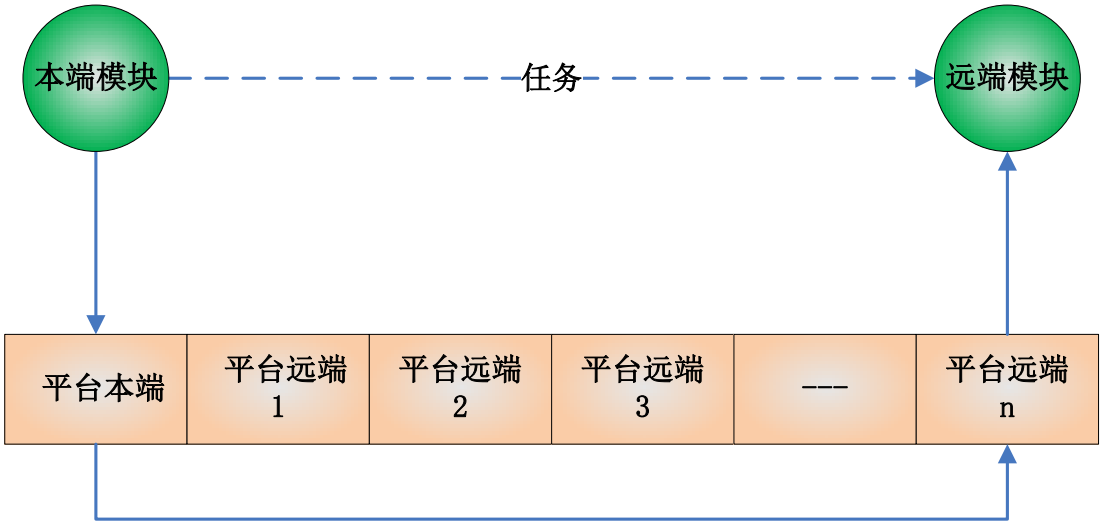
4、 每个物理主机可以部署一个或多个平台，每个平台对应一个通信子，每个通信子包含若干进程，每个进程可以实例化多个模块，每个进程可以开启多个线程（或协程)。

线程（或协程）对用户透明，用户应用不依赖于具体线程（或协程）数，当受限于线程（或协程）数。

5、 一个模块管理者是管理多个模块的模块资源池。模块和模块管理者对用户可见。

6、 部署在不同物理主机上的全部平台构成一个大的、逻辑的平台，任务在平台中寻址。

7、 平台由任务驱动，即平台之间交换数据或进行逻辑控制的载体是任务。通信过程中，发起任务的为平台本端，接收任务的为平台远端；发送任务的模块为本端模块，接收任务的模块为远端模块。如图所示。



图例说明：

本端模块向远端模块发送任务。虚线为逻辑路线，实线为传输路线（寻址路线）。

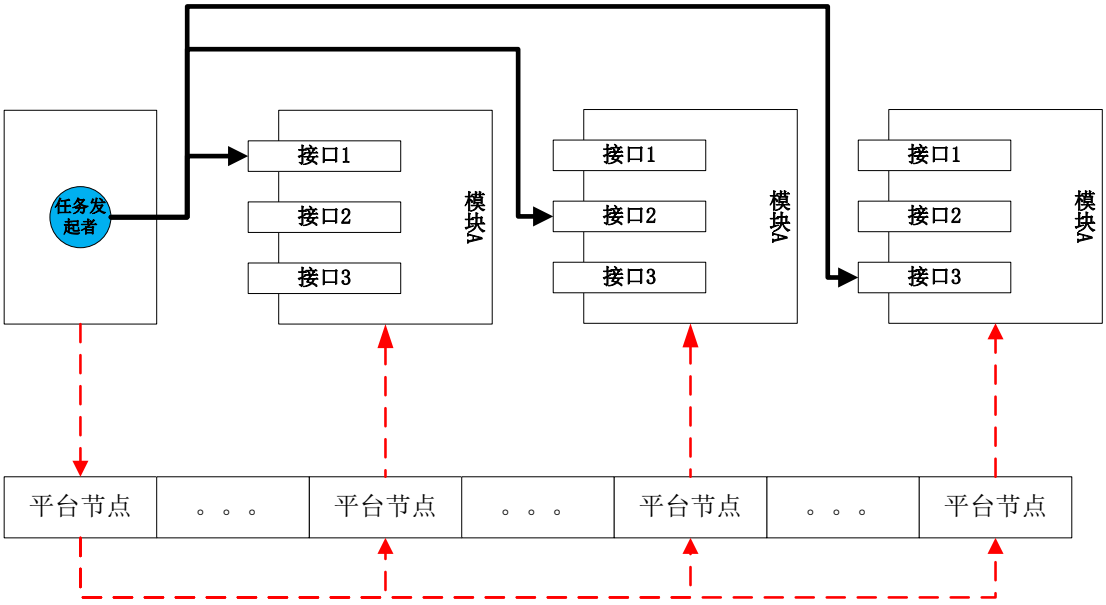
注：任务发起者可以是平台模块，也可以是任意用户应用。

图例说明：

远端模块与本端模块可以是同一个模块，也可以是不同的模块，即允许模块向自身发送任务。

图例说明：

平台远端与平台本端可以是同一个平台，也可以是不同平台，即允许平台向自身发送任务。

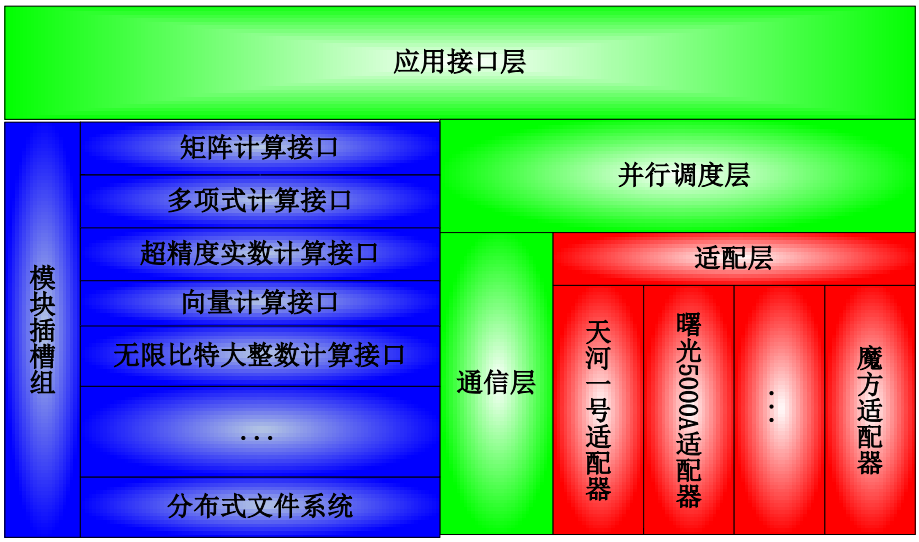


图例说明：

8、 任务发起者发起一次任务，包含三个任务请求，分别调用三个平台节点的模块 A 的三个接口。黑色实线为任务的逻辑路线，红色虚线为任务的传输路线（寻址路线）。

6、 平台架构

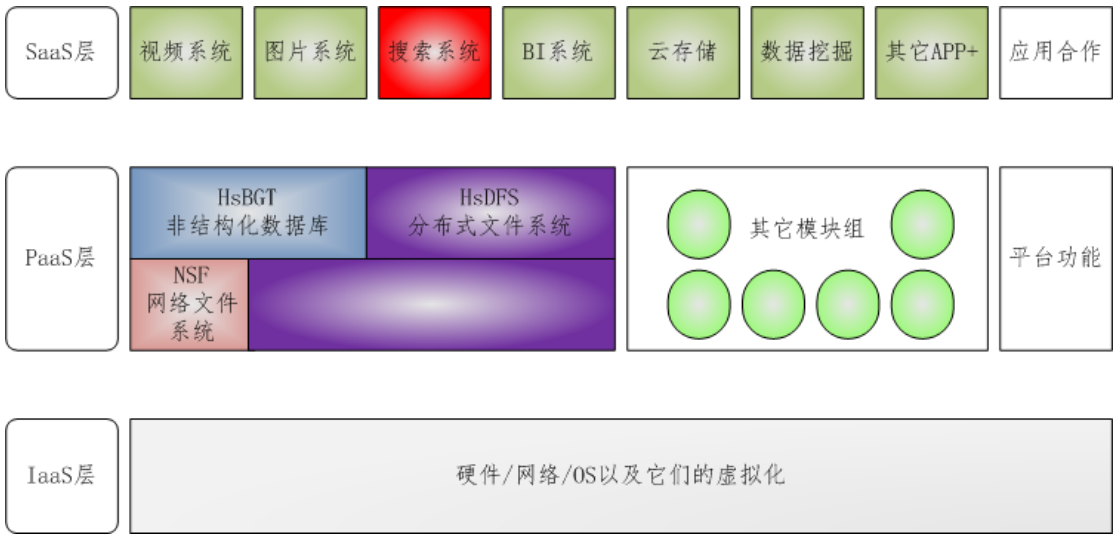
6.1 平台架构如图所示



图例说明：

- 应用接口层：平台面向应用软件提供的接口集
- 模块插槽组：提供各种功能模块的接口集，比如矩阵运算、分布式文件系统、分词检索等。支持无限扩展。
- 并行调度层：平台在进程或模块间实行负载均衡的功能集
- 通　信　层：基于 SOCKET 技术的通信功能集，支持任务转发、任务路由等寻址技术。
- 适　配　层：平台对上层应用软件完全屏蔽掉底层环境差异的功能集

6.2 应用架构如图所示

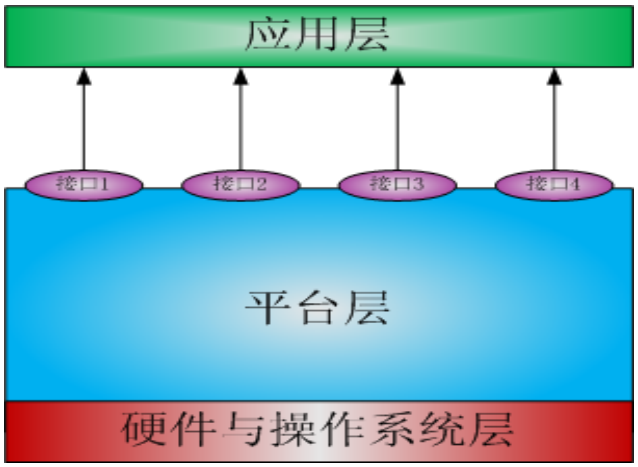


7、 平台特性

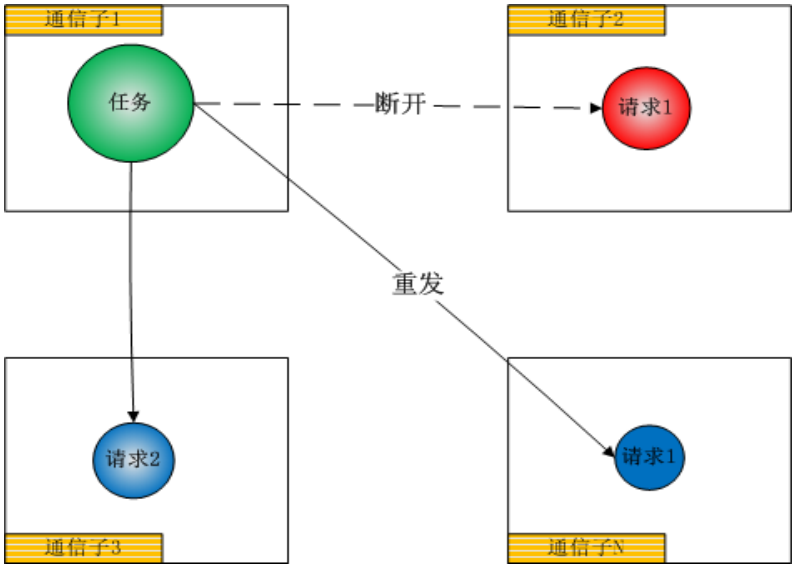
平台具有如下特性：

- **高层抽象**

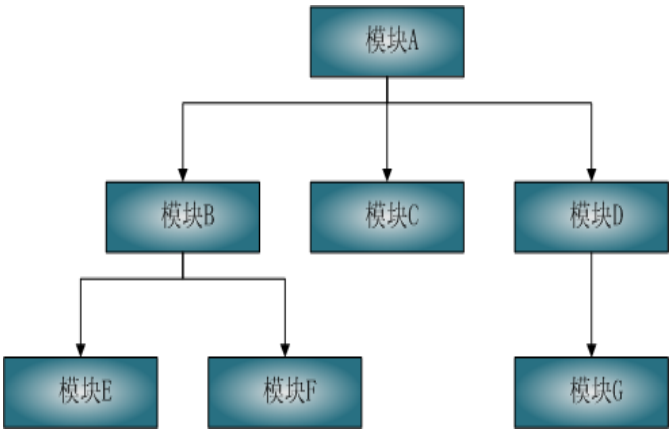
为平台用户屏蔽底层硬件结构差异与复杂的底层通信管理，提供**一致的应用编程接口**以及并行任务的自动指派和调度策略，达到通用、高效和**透明**目的。



- **可容错性、可扩展性、可移植性、可重用性、可继承性、可维护性**
  基于**模块技术**、**容错技术**、负载均衡技术等，为平台用户提供强大而丰富的特性



图例说明：通信子 1 发起一次任务包含两个任务请求，请求 1 发往通信子 2，请求 2 发往通信子 3。由于通信故障，通信子 1 与通信子 2 失去联系，通信子 1 将请求 1 重新调度，发往某个负载较轻的通信子 N。

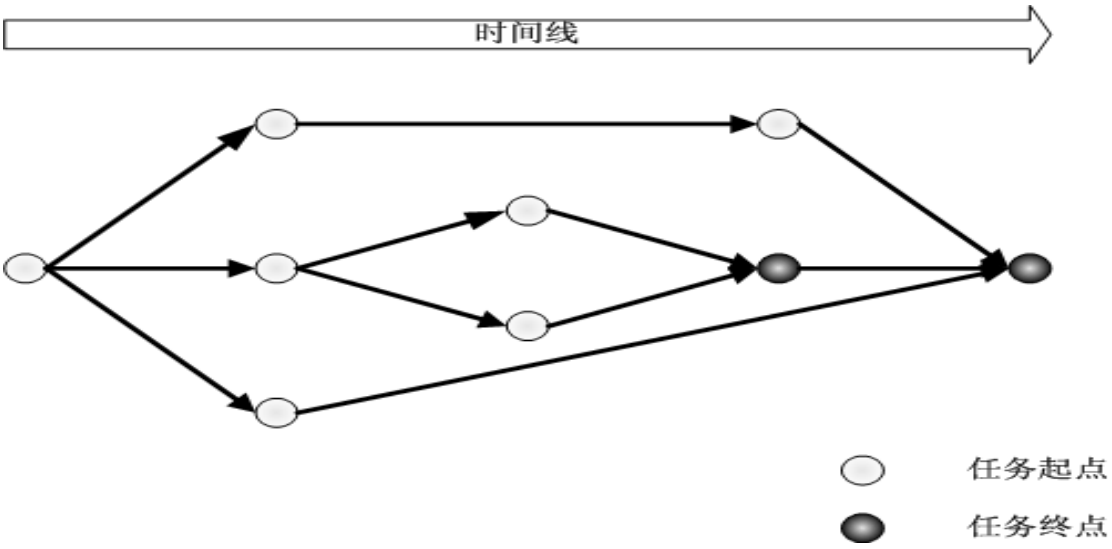


图例说明：

模块技术为平台享有完全自主知识产权的技术之一。模块支持继承性。如果将模块比如成砖块，继承性则允许将这些砖块堆成摩天大厦。

- **并行复杂逻辑控制**

基于**任务驱动技术**与负载均衡技术，以模块间通信为支撑，为平台用户提供任务的动态创建、自动指派、智能调度。**任务同步技术**支持复杂并行逻辑控制，奠定大型并行应用软件的逻辑控制基础。

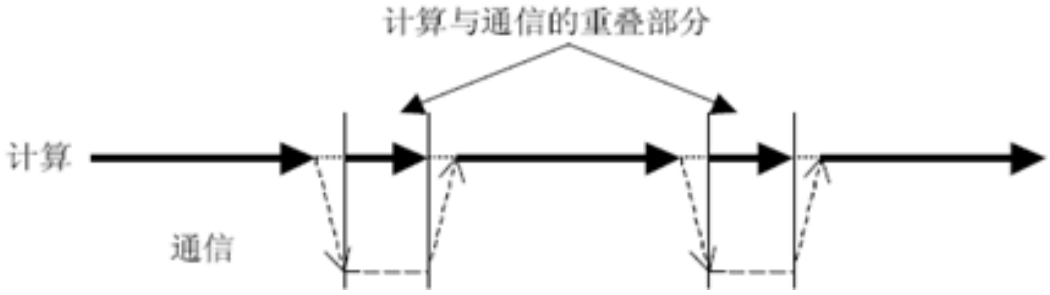


图例说明：

一个任务包含多个子任务，子任务在远端可以分裂出多个新的子任务，从而形成一棵任务树。用户的应用，或者实际生活中的并行场景，往往面临着复杂的并行分裂与汇聚，平台支持复杂逻辑控制，用户只须关心应用层的业务逻辑，而不必关心底层的逻辑控制。

- **通信与计算的时间重叠**

基于**无阻塞通信技术**，实现通讯与计算在时间上的重叠，用户无须考虑任务何时发起，响应何时收到，解放了并行软件的设计模式，有利于构建大规模并行计算软件

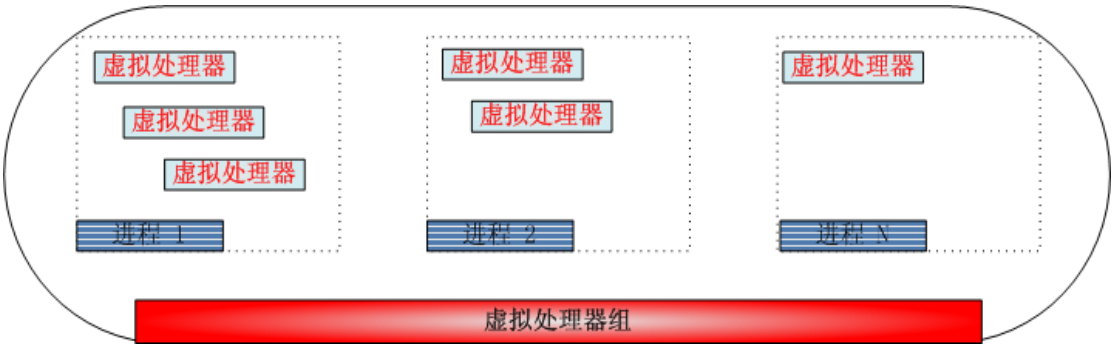


图例说明：

在并行环境中，用户发起并行任务往往是条件触发、或随机触发，即任务发起的时点不确定。平台的无阻塞通信技术可以解决任务偶发问题，大大解放了并行设计模式。同时，无阻塞通信技术意味着计算与通信可以在时间上重叠，从而提高系统效率。

- **处理器虚拟化技术**

处理器虚拟化技术为平台用户提供了无限的处理器资源，无论最终实际处理器的数量如何，平台保证并行应用软件的完美运行

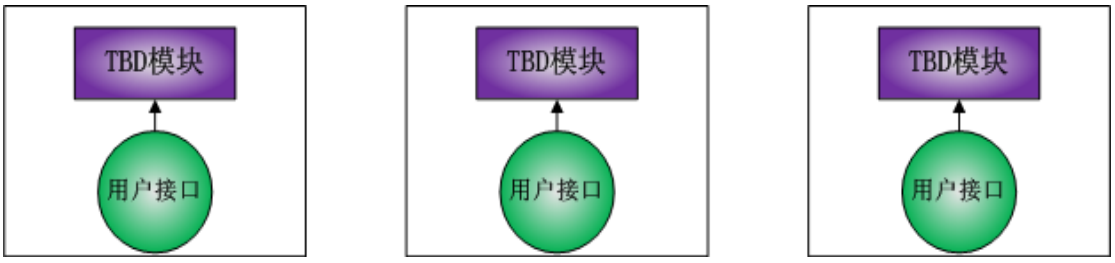


图例说明：

虚拟处理器位于各进程中，全部进程的虚拟处理器构成一个庞大的虚拟处理器池，应用根据需要，结合负载均衡技术和容错技术，使用虚拟处理器资源。

- **用户接口兼容技术**

基于用户接口兼容技术，用户已有的串行接口函数可直接**被并行化**，自动享有平台提供的全部并行特性。

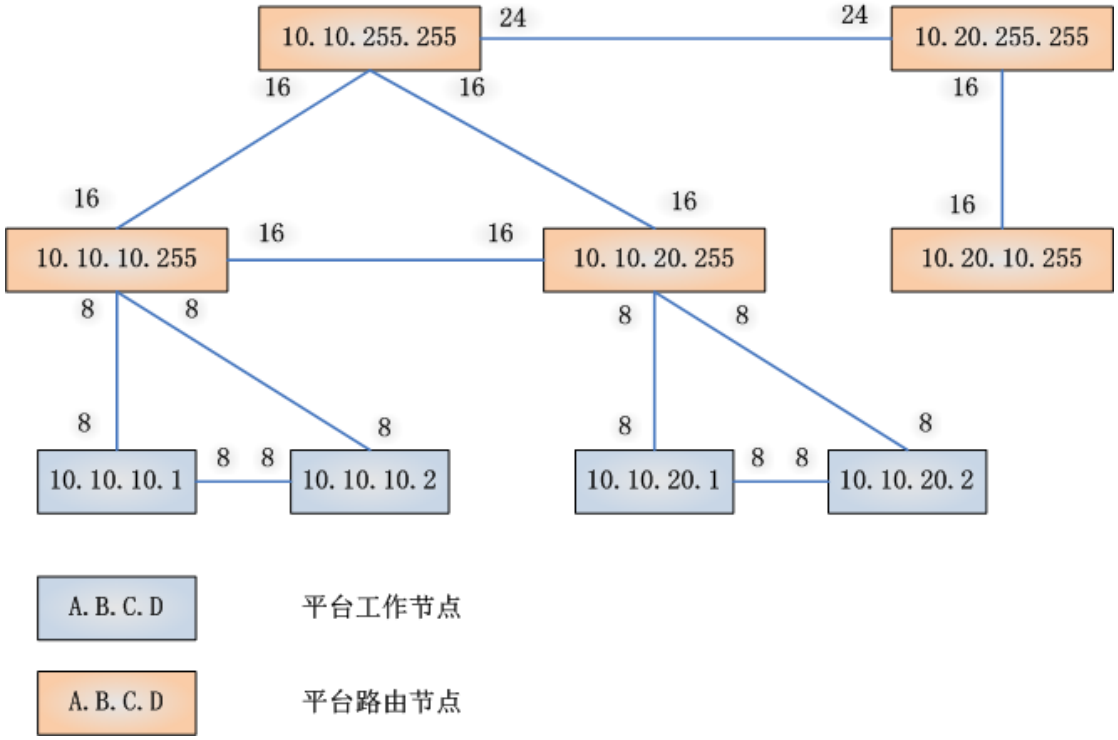


图例说明：

用户现有代码量可能很庞大，改造成平台模块的成本较大，或者用户现有应用适合数据切分后小块数据的处理，这时可以使用平台的用户接口兼容技术直接并行化。平台提供一个 TBD 模块加载用户接口，通过 TBD 模块向用户接口输入数据，或者，进一步，通过 TBD 模块实现不同实例的用户接口之间的数据交换。

- **业务层组网技术**

业务层可以独立于传输层网路（比如 IP 网络），独立组成一张业务层网络，每个业务层网络平台节点为一个通信子，以类 IP 地址的点分形式标识。用户根据业务需求特点，可以划分不同的业务子网，实现资源裂分。



图例说明：

比如平台节点 10.10.10.1 发送任务到 10.10.20.1，则任务传输路径是 10.10.10.10.1 -> 10.10.10.255 -> 10.10.20.255 -> 10.10.20.1

如果 10.10.10.255 与 10.10.20.255 断连，则任务传输路径可以是 10.10.10.10.1 -> 10.10.10.255 -> 10.10.255.255 -> 10.10.20.255 -> 10.10.20.1

平台节点标识完全独立于物理承载网络的 IP 地址，任务寻址也完全独立于 IP 层寻址。

8、 平台关键技术

- **处理器虚拟化技术**

优点：处理器虚拟化技术为平台用户提供了无限的处理器资源，无论最终实际处理器的数量如何，平台保证并行应用软件的完美运行

- **模块调度技术**

优点：突破传统以进程为单元的调度模式，可调度的模块资源可视为无限。

系统所支持的进程数量是有限的，传统并行计算要考虑起多少进程并对其进行调度，在软件设计上有诸多局限，比如，系统不支持开指定数量的进程可能导致软件无法正常运行。

模块调度纵向上是进程内部的调度，引入线程（或协程）后成为进程内部的并行调度；横向上是进程间的并行调度。

平台不以进程为最小调度单元，而以模块为最小调度单元，模块的数量可视为无限。这样，当应用软件设计以模块为调度单元时，不必考虑模块数量的限制，更不必考虑开了多少进程，提高并行设计模式的抽象层次，比如，可以在少量 CPU 的环境中模拟大量 CPU 环境中的计算过程。

- **任务驱动技术**

优点：部署并行计算平台的各节点通过任务触发的方式启动计算，反馈计算结果。各节点是惰性的，被动接受任务，促使了在软件设计上分离任务收发和算法实现。支持三种优先级任务：抢占级、高级、正常级。

- **复杂并行逻辑控制技术**

优点：平台任务同步技术支持复杂并行逻辑控制，奠定大型并行应用软件的逻辑控制基础。

- **容错技术**

优点：相对于 MPI 进程 “同生共死”的特点，即一个通信子内的一个进程出错退出时会导致该通信子的所有进程退出，平台通过动态增减管理多个从通信子。一个或几个从通信子受损不影响其他通信子的正常工作，同时发往受损通信子的任务被重新调度至正常工作的通信子，从而实现平台的容错。

- **负载均衡技术**

1. 轮询负载均衡：轮流向各模块分配任务
2. 进程负载均衡：根据进程的负载状况分配任务，保持进程间的负载均衡
3. 模块负载均衡：根据模块的负载状况分配任务，保持模块间的负载均衡

优点：运行期间动态负载均衡，有效利用空闲计算资源

- **串并程序转换技术**

优点：目前任务支持函数级，高于数据并行。将串行程序改写为并行程序简单，反之亦然。

用户已有的串行接口函数，可不修改或少量修改接口定义，即可自动获得平台全部并行特性。

- **模块插槽组技术**

优点：模块是一组数据与操作的集合。模块数量可视为无限。模块之间可以存在继承关系。模块运行是启停式。同一个模块注入不同的参数，生成不同的实例对象提供服务。模块间存在继承关系，模块组内部形成分层结构，具有可扩展、可优化、可调度的特性。

- **异步无阻塞通讯技术**

优点：实现通讯与计算在时间上的重叠；不必考虑任务何时发出，响应何时收到，解放了并行计算软件的设计模式，有利于构建大规模并行应用软件。

- **任务自动寻址技术**

优点：任务根据平台业务层组网配置，经过软体交换机和软体路由器，自动寻址到目的模块。

- **业务层独立组网技术**

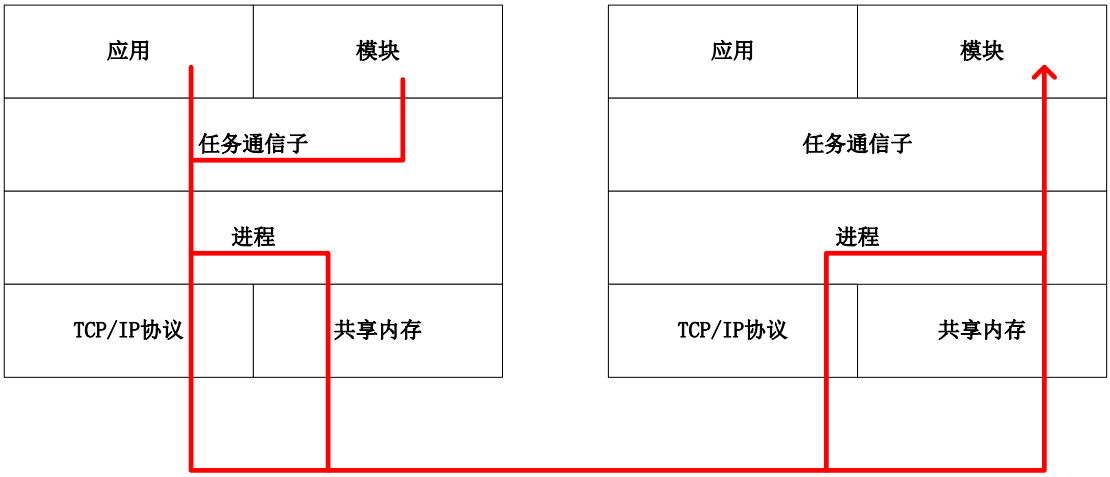
优点：业务层独立于传输层网路（比如 IP 网络），独立组成一张业务层网络，每个网络节点为一个通信子，以类 IP 地址的点分形式标识。用户根据业务需求特点，可以划分不同的业务子网，实现资源裂分。

- **任务超时技术**

优点：任务超时技术有利于用户建立业务层时序体系。

9、 平台通信协议

9.1 任务通信协议栈



图例说明：

✓ 任务发起者可以是应用，也可以是模块

✓ 同一个任务通信子中的两个进程之间通信走共享内存，不同任务通信子的两个进程之间通信走 TCP/IP 协议。

9.2 任务通信语义

平台由任务驱动，一个任务包含一个或多个子任务。任务通信语义表达的是，任务提交给模块管理者，由平台自动完成负载均衡、任务分发、任务路由、任务重调度、任务超时、任务执行以及任务反馈。

9.3 任务通信支持广播、多播和单播、点到点

广播：向一个模块管理者中的全部模块发起一个任务
多播：向一个模块管理者中的若干个模块发起一个任务
单播：向一个模块管理者中的某个模块发起一个任务
点到点：向一个特定的远端模块发起一个任务

9.4 任务策略

- 任务触发遵循模块的启停模式，
  1. 激活（或启动）远端模块
  2. 提交或接收任务序列并处理
  3. 去激活（或停止）远端模块

- 任务提交策略支持
  1. 单函数单任务（SISD），即单次任务中包含一个函数一次调用，发往一个远端。
  2. 单函数多任务（SIMD），即单次任务中包含一个函数多次调用，发往多个远端。
  3. 多函数多任务（MIMD），即单次任务包含多个函数多次调用，发往多个远端。

- 任务结束条件
  1. 如果任务请求需要全部响应，则任务发起者收齐全部响应视为任务结束
  2. 如果任务请求需要部分响应，则任务发起者收齐指定数目的响应视为任务结束
  3. 如果任务请求不需要响应，则任务发起者提交任务完毕视为任务结束
  4. 如果任务请求需要响应，但等待响应超时，则视为任务结束

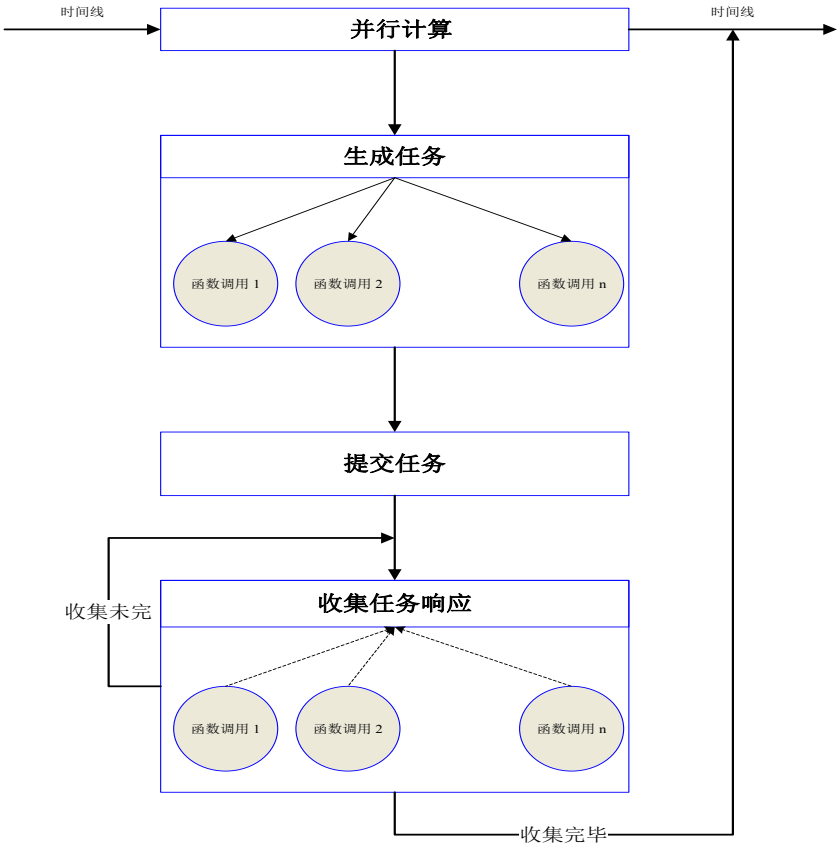
- 任务优先级由低到高，
  1. 正常优先级
  2. 高优先级
  3. 抢占优先级

- 任务优先级策略
  1. 优先级高的任务先于优先级低的任务被调度
  2. 同属正常优先级的任务遵循先进先出原则（FIFO）
  3. 同属高优先级的任务遵循先进先出原则（FIFO）
  4. 同属抢占优先级的任务遵循后进先出原则（LIFO）

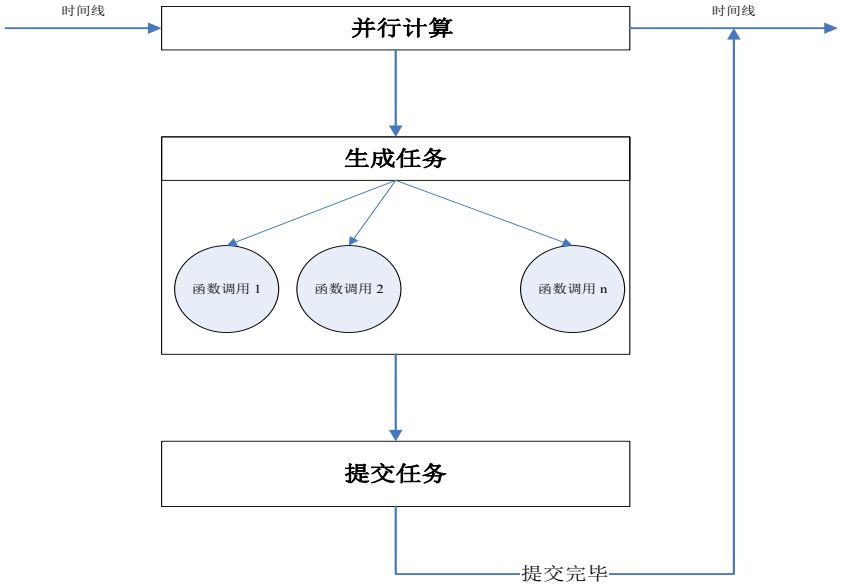
10、平台任务提交流程

1. 用户应用在顺序执行过程中插入并行计算

2. 用户应用生成一个任务，含多个任务请求，每个任务请求是一次函数调用
3. 用户应用向系统提交任务，任务被调度时系统根据负载均衡策略决定任务的分发
4. 对有响应的任务，用户应用等待并收集任务响应 ，直至收齐响应
5. 对无响应的任务，用户应用等待任务提交完毕
6. 用户应用回到顺序执行过程中继续，任务结束



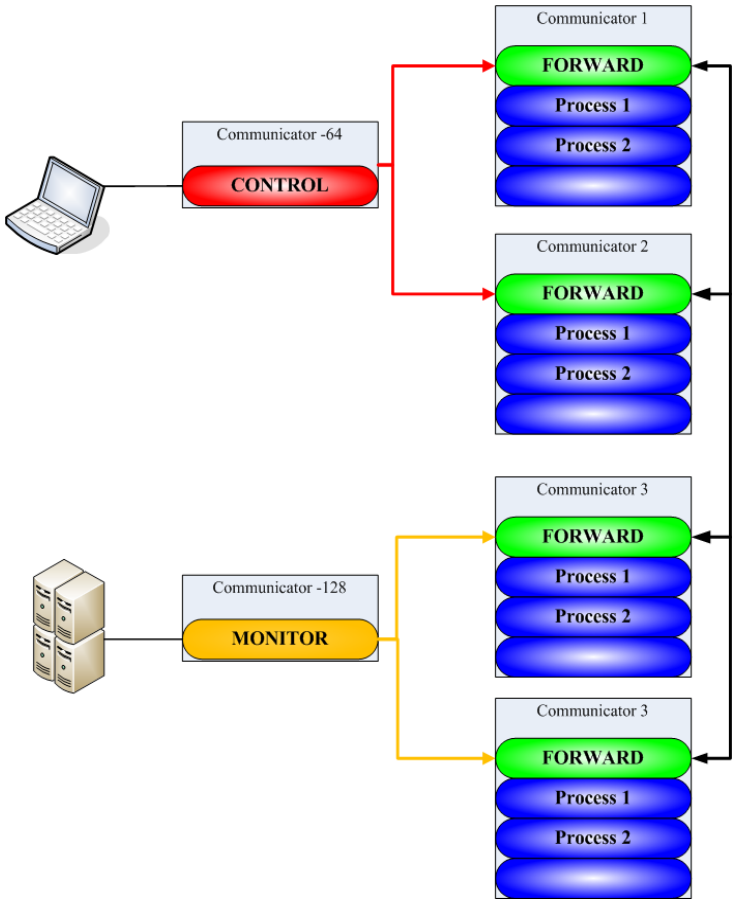
任务提交图（有响应）



任务提交图（无响应）

11、平台模块管理

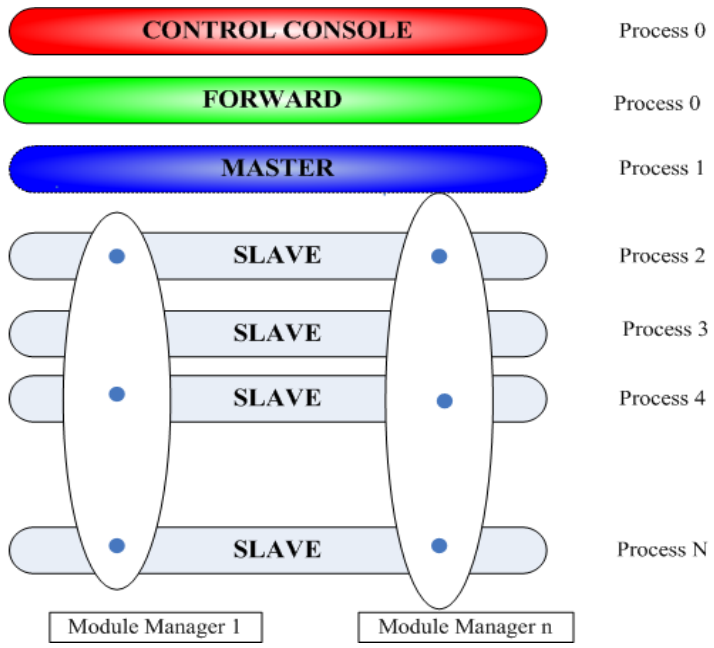
11.1 进程管理



图例说明：

- 进程存在于通信子
- 通信子分工作通信子、调测通信子和监控通信子三类
- 调测通信子标识从 0.0.0.64 到 0.0.0.95，监控通信子标识从 0.0.0.96 到 0.0.0.127，其余非零值均可作为工作通信子标识。
- 发起任务的通信子为主通信子，其他为从通信子。主从通信子是相对的概念。
- 调测通信子的 0 号进程为控制进程，用作调测和操作维护。建议调测通信子包含且仅包含一个进程。
- 监控通信子的 0 号进程为监控进程，用作监控信息采集。建议监控通信子包含且仅包含一个进程。
- 每个工作通信子的 0 号进程为任务转发（中继）进程
- 发起任务的进程为主进程，其他进程为从进程

11.2 模块管理



图例说明：

- 全部通信子的全部进程构成进程资源池
- 模块存在于进程
- 每个进程可创建多个模块
- 模块管理者管理若干进程的若干模块
- 模块管理者管理的模块数量因需而定
- 模块管理者跨越的进程分布和通信子分布由模块分布决定
- 模块服从 1:N 主从关系模型
- 主模块可以接受模块管理者管理
- 主模块可以存在于任何进程
- 从模块可以存在于任何进程
- 主从模块可以是同一个模块
- 主从模块可以存在于同一个进程
- 一个模块至多能被一个管理者管理
- 支持基于模块的负载平衡
- 模块间服从任务请求与响应的通信模型
- 模块之间可以存在继承关系
- 模块间不共享内存
- 模块通过启停模式创建和销毁模块实例

12、平台相关限定

1、 平台在异构环境中部署时，禁止 32 位操作系统和 64 位操作系统异构的情形

2、 平台支持的基本数据类型约定

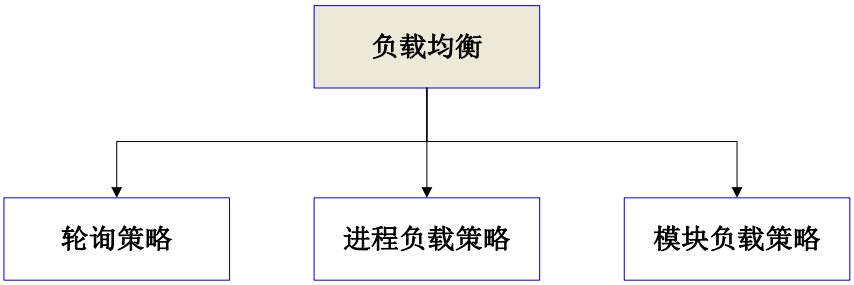
类型名称	字节数	备注
UINT32	4/8 字节	一个字长，即 32 位系统上为 4 字节（32 比特），64 位系统上为 8 字节（64 比特）
UINT16	2 字节	
UINT8	1 字节	
REAL	8 字节	双精度浮点数，double 型
uint32_t	4 字节	
uint16_t	2 字节	
uint8_t	1 字节	
word_t	4/8 字节	一个字长，即 32 位系统上为 4 字节（32 比特），64 位系统上为 8 字节（64 比特）

3、 平台禁止使用枚举类型，常值定义建议用 define 方式定义并明示类型

4、 用户定义任务时若出现常值，必须是 UINT32 类型，即一个机器字长，并明示

13、平台负载均衡

平台支持轮询、进程负载、模块负载等三种负载均衡策略。



图例说明：

- 轮询策略

  轮流向模块管理者中的模块分派任务请求

- 进程负载策略

  根据进程总的负载状况，选择模块管理者所跨越的进程中负载最小的进程，向其中的某个模块分派任务

- 模块负载策略

  根据模块管理者中模块的负载状况，向负载最小的模块分派任务

14、代码举例

图例说明：

这是做并行矩阵乘演示的一段代码。嵌套的两个循环完成两个矩阵行列块的乘法以及累加。这里的并行计算用到平台的三个接口：

- task_new： 创建任务管理者。即创建一个任务模板，参数表中包含模块管理者（即模块资源池）、任务优先级、是否需要任务响应以及需要的任务响应数量
- task_inc：向任务管理者中添加子任务请求。每个任务请求是一次函数调用。参数表中包含任务管理者、函数返回值、函数参数表。
- task_wait：任务阻塞，即向平台提交被创建的任务（管理者），参数表中包含任务管理者、任务生命周期、任务是否接受重调度（容错机制）、任务响应检查函数指针。

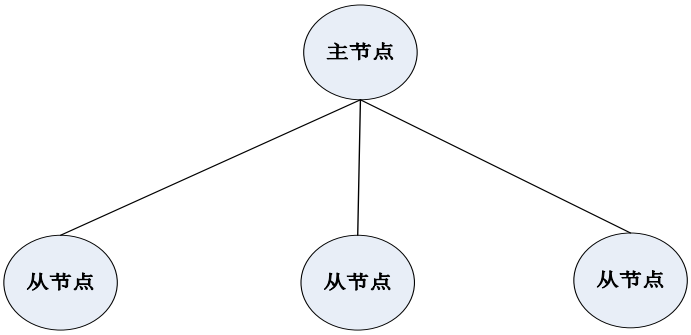
如果去掉平台的三个接口，将 task_inc 中的函数调用释放出来，还原成普通的函数调用，本段代码即为串行代码。

15、平台节点组网模型

平台节点组网形成应用所需的集群。平台目前支持主从组网、全连组网两种基本模型，以及方便分布式文件系统 HsDFS 的组网模型，NoSQL 分布式数据库 HsBGT 的组网模型。

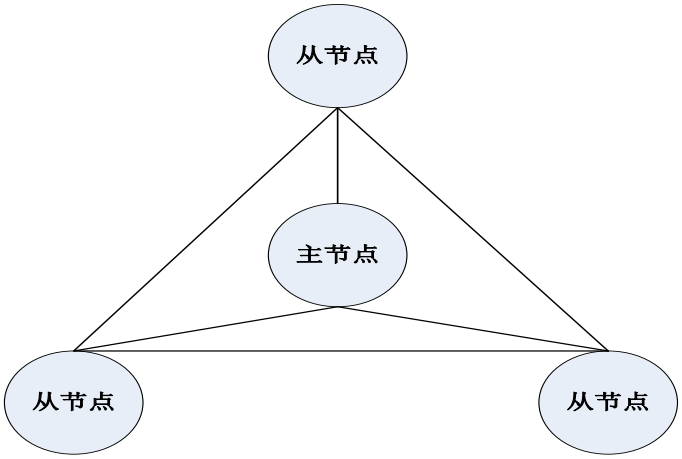
15.1 主从组网模型

一个主节点，多个从节点，主节点与所有从节点建立连接，从节点之间无需连接。



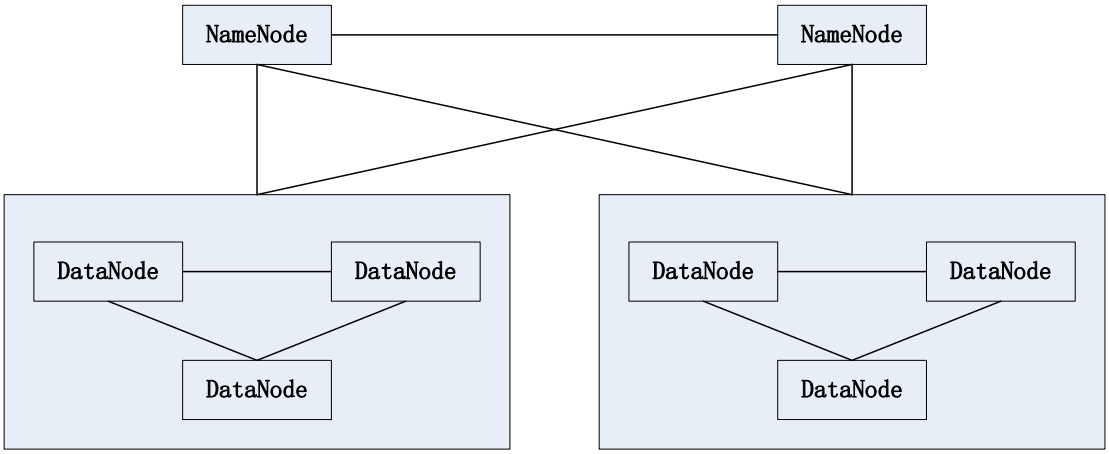
15.2 全连组网模型

若干节点彼此全连接。



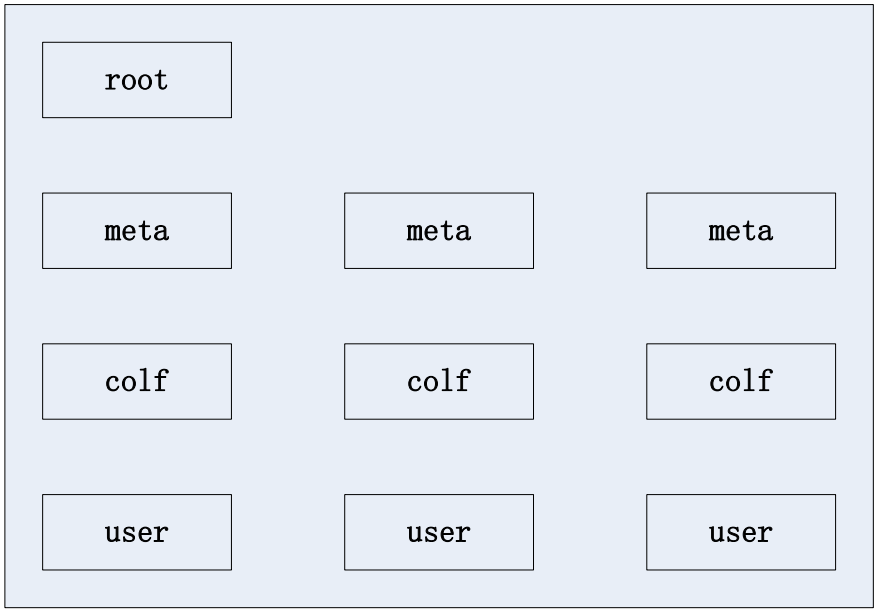
15.3 HsDFS 组网模型

两个 NameNode 节点全连，每三个 DataNode 一组全连，NameNode 与 DataNode 全连，不同的 DataNode 组之间无需连接。



15.4 HsBGT 组网模型

一个或多个节点用作 root table server 并标识出来，其余节点为 meta table server、或 colf table server、或 user table server，无需标识出来。table serve 之间全连。



16、平台配置文件说明

每个平台节点需要一个配置文件，已引导平台节点的启动和集群组网。配置文件名为 config.xml，由三部分组成：平台节点到物理节点的映射、平台节点所在集群的组网形式、平台节点各进程中的线程池配置参数。(【注】这里暂忽略业务层组网所需的软路由配置部分)

配置举例

```
<sysConfig>
```

```
<taskConfig>
    <tasks tcid="10.10.10.1"    maski="0" maske="16" srvipaddr="192.168.231.18" srvport="57001" cluster="1"/>
    <tasks tcid="10.10.20.1"    maski="0" maske="16" srvipaddr="192.168.231.78" srvport="57201" cluster="1"/>
    <tasks tcid="10.10.10.2"    maski="0" maske="16" srvipaddr="192.168.231.18" srvport="57002" cluster="1"/>
    <tasks tcid="10.10.10.3"    maski="0" maske="16" srvipaddr="192.168.231.38" srvport="57003" cluster="1"/>
    <tasks tcid="10.10.10.4"    maski="0" maske="16" srvipaddr="192.168.231.48" srvport="57004" cluster="1"/>
    <tasks tcid="10.10.10.5"    maski="0" maske="16" srvipaddr="192.168.231.58" srvport="57005" cluster="1"/>
    <tasks tcid="10.10.10.6"    maski="0" maske="16" srvipaddr="192.168.231.78" srvport="57006" cluster="1"/>
    <tasks tcid="10.10.10.7"    maski="0" maske="16" srvipaddr="192.168.231.58" srvport="57007" cluster="1"/>
    <tasks tcid="10.10.10.8"    maski="0" maske="16" srvipaddr="192.168.231.78" srvport="57008" cluster="1"/>
    <tasks tcid="10.10.10.91" maski="0" maske="16" srvipaddr="192.168.231.58" srvport="57009" cluster="1"/>
    <tasks tcid="10.10.10.92" maski="0" maske="16" srvipaddr="192.168.231.78" srvport="57010" cluster="1"/>
    <tasks tcid="10.10.10.101" maski="0" maske="16" srvipaddr="192.168.231.58" srvport="57011" cluster="1"/>
    <tasks tcid="10.10.10.102" maski="0" maske="16" srvipaddr="192.168.231.78" srvport="57012" cluster="1"/>
    <tasks tcid="10.10.10.201" maski="0" maske="16" srvipaddr="192.168.231.58" srvport="57013" cluster="1"/>
    <tasks tcid="10.10.10.202" maski="0" maske="16" srvipaddr="192.168.231.78" srvport="57014" cluster="1"/>
    <tasks tcid="10.10.30.1"    maski="0" maske="16" srvipaddr="192.168.231.48" srvport="57301" cluster="1,2"/>
    <tasks tcid="10.10.30.2"    maski="0" maske="16" srvipaddr="192.168.231.58" srvport="57302" cluster="1,2"/>
    <tasks tcid="10.10.30.3"    maski="0" maske="16" srvipaddr="192.168.231.58" srvport="57303" cluster="1,2"/>
    <tasks tcid="10.10.30.4"    maski="0" maske="16" srvipaddr="192.168.231.78" srvport="57304" cluster="1,2"/>
    <tasks tcid="10.10.30.5"    maski="0" maske="16" srvipaddr="192.168.231.48" srvport="57305" cluster="1,2"/>
    <tasks tcid="10.10.30.7"    maski="0" maske="16" srvipaddr="192.168.231.78" srvport="57307" cluster="1,2"/>

    <tasks tcid="0.0.0.64" maski="32" maske="32" srvipaddr="192.168.231.18" srvport="57064" cluster="3"/>
</taskConfig>

<clusters>
    <cluster id="1" name="hsdfs_01" model="hsdfs">
        <node role="namenode" tcid="10.10.10.1"     rank="0" npdir="/home/ezhocha/hsdfs/10.10.10.1"/>
        <node role="namenode" tcid="10.10.20.1"     rank="0" npdir="/home/ezhocha/hsdfs/10.10.20.1"/>
        <node role="datanode" tcid="10.10.10.2"     rank="0" dndir="/home/ezhocha/hsdfs/10.10.10.2" group="dn_grp_01"/>
        <node role="datanode" tcid="10.10.10.3"     rank="0" dndir="/home/ezhocha/hsdfs/10.10.10.3" group="dn_grp_01"/>
        <node role="datanode" tcid="10.10.10.4"     rank="0" dndir="/home/ezhocha/hsdfs/10.10.10.4" group="dn_grp_01"/>
        <node role="client"     tcid="10.10.10.5"     rank="0"/>
        <node role="client"     tcid="10.10.10.6"     rank="0"/>
        <node role="client"     tcid="10.10.10.7"     rank="0"/>
        <node role="client"     tcid="10.10.10.8"     rank="0"/>
        <node role="client"     tcid="10.10.10.91"    rank="0"/>
        <node role="client"     tcid="10.10.10.92"    rank="0"/>
        <node role="client"     tcid="10.10.10.101" rank="0"/>
        <node role="client"     tcid="10.10.10.102" rank="0"/>
        <node role="client"     tcid="10.10.10.201" rank="0"/>
        <node role="client"     tcid="10.10.10.202" rank="0"/>
        <node role="client"     tcid="10.10.30.1" rank="0"/>
```

```
        <node role="client"     tcid="10.10.30.2" rank="0"/>
        <node role="client"     tcid="10.10.30.3" rank="0"/>
        <node role="client"     tcid="10.10.30.4" rank="0"/>
        <node role="client"     tcid="10.10.30.5" rank="0"/>
        <node role="client"     tcid="10.10.30.7" rank="0"/>
    </cluster>
    <cluster id="2" name="hsbgt_01" model="hsbgt" roottabledir="/home/ezhocha/hsbgt">
        <node role="table" tcid="10.10.30.5"     rank="0" group="root"/>
        <node role="table" tcid="10.10.30.7"     rank="0" />
        <node role="table" tcid="10.10.30.1"     rank="0"/>
        <node role="table" tcid="10.10.30.2"     rank="0"/>
        <node role="table" tcid="10.10.30.3"     rank="0"/>
        <node role="table" tcid="10.10.30.4"     rank="0"/>
    </cluster>
    <cluster id="3" name="debug" model="master_slave">
        <node role="master" tcid="0.0.0.64"     rank="0"/>
        <node role="slave" tcid="10.10.10.1"     rank="0"/>
        <node role="slave" tcid="10.10.20.1"     rank="0"/>
        <node role="slave" tcid="10.10.10.2"     rank="0"/>
        <node role="slave" tcid="10.10.10.3"     rank="0"/>
        <node role="slave" tcid="10.10.10.4"     rank="0"/>
        <node role="slave" tcid="10.10.10.5"     rank="0"/>
        <node role="slave" tcid="10.10.10.6"     rank="0"/>
        <node role="slave" tcid="10.10.10.7"     rank="0"/>
        <node role="slave" tcid="10.10.10.8"     rank="0"/>
        <node role="slave"     tcid="10.10.10.91"    rank="0"/>
        <node role="slave"     tcid="10.10.10.92"    rank="0"/>
        <node role="slave"     tcid="10.10.10.101" rank="0"/>
        <node role="slave"     tcid="10.10.10.102" rank="0"/>
        <node role="slave"     tcid="10.10.10.201" rank="0"/>
        <node role="slave"     tcid="10.10.10.202" rank="0"/>
        <node role="slave" tcid="10.10.30.1"     rank="0"/>
        <node role="slave" tcid="10.10.30.2"     rank="0"/>
        <node role="slave" tcid="10.10.30.3"     rank="0"/>
        <node role="slave" tcid="10.10.30.4"     rank="0"/>
        <node role="slave" tcid="10.10.30.5"     rank="0"/>
        <node role="slave" tcid="10.10.30.7"     rank="0"/>
    </cluster>
</clusters>
<parasConfig>
    <paraConfig tcid="10.10.10.1" rank="0">
        <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
    </paraConfig>
```

```
<paraConfig tcid="10.10.20.1" rank="0">
    <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
</paraConfig>

<paraConfig tcid="10.10.10.2" rank="0">
    <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
</paraConfig>

<paraConfig tcid="10.10.10.3" rank="0">
    <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
</paraConfig>

<paraConfig tcid="10.10.10.4" rank="0">
    <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
</paraConfig>

<paraConfig tcid="10.10.10.5" rank="0">
    <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
</paraConfig>

<paraConfig tcid="10.10.10.6" rank="0">
    <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
</paraConfig>

<paraConfig tcid="10.10.10.7" rank="0">
    <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
</paraConfig>

<paraConfig tcid="10.10.10.8" rank="0">
    <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
</paraConfig>

<paraConfig tcid="10.10.10.91" rank="0">
    <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
</paraConfig>

<paraConfig tcid="10.10.10.92" rank="0">
    <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
</paraConfig>

<paraConfig tcid="10.10.10.101" rank="0">
    <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
</paraConfig>
```

```
        <paraConfig tcid="10.10.10.102" rank="0">
            <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
        </paraConfig>

        <paraConfig tcid="10.10.10.201" rank="0">
            <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
        </paraConfig>

        <paraConfig tcid="10.10.10.202" rank="0">
            <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
        </paraConfig>

        <paraConfig tcid="10.10.30.1" rank="0">
            <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
        </paraConfig>

        <paraConfig tcid="10.10.30.2" rank="0">
            <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
        </paraConfig>

        <paraConfig tcid="10.10.30.3" rank="0">
            <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
        </paraConfig>

        <paraConfig tcid="10.10.30.4" rank="0">
            <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
        </paraConfig>

        <paraConfig tcid="10.10.30.5" rank="0">
            <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
        </paraConfig>

        <paraConfig tcid="10.10.30.7" rank="0">
            <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
        </paraConfig>

        <paraConfig tcid="0.0.0.64" rank="0">
            <threadConfig maxReqThreadNum="4" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
        </paraConfig>
    </parasConfig>
</sysConfig>
```

关于上面配置的解释：

```
<tasks tcid="10.10.10.1"  maski="0" maske="16" srvipaddr="192.168.231.18" srvport="57001"
```

```
cluster="1"/>
```

该行配置表示，平台节点 10.10.10.1 的内网掩码为 0.0.0.0，外网掩码为 255.255.0.0，绑定到 IP 地址为 192.168.231.18 的网卡和端口号 57001，并且该平台节点隶属于集群 1。

```
<cluster id="1" name="hsdfs_01" model="hsdfs">
```

该行配置标识，集群 1 为 HSDFS 组网模型。

```
<paraConfig tcid="10.10.10.1" rank="0">
  <threadConfig maxReqThreadNum="256" maxRspThreadNum="32" maxStackSize="65536" stackGuardSize="4096"/>
</paraConfig>
```

该段配置标识平台节点 10.10.10.1 中的第 1 个进程（进程编号从 0 开始）的线程配置为：任务请求处理线程（或协程）最多为 256 个，任务响应处理线程（或协程）最多为 32 个，线程（或协程）堆栈最大为 64KB，其中线程堆栈保护段大小为 4KB

17、平台节点起停

平台节点启动可通过命令行直接运行，比如

```
./tbd -np 2 -tcid 10.10.10.1
```

表示启动平台节点 10.10.10.1，该平台节点有 2 个进程

平台节点停止在不影响业务的前提下，可通过调测口执行 shutdown 命令，比如

```
bgn> shutdown work tcid 10.10.10.1
```

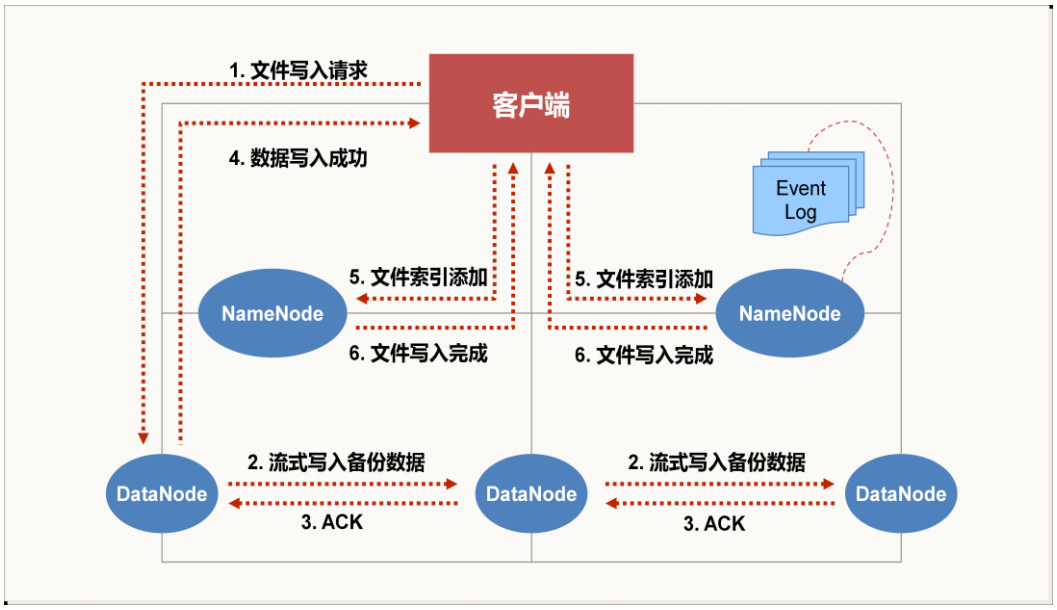
表示停止平台节点 10.10.10.1

其中调测口的启动命令形如

```
./tbd -tcid 0.0.0.64
```

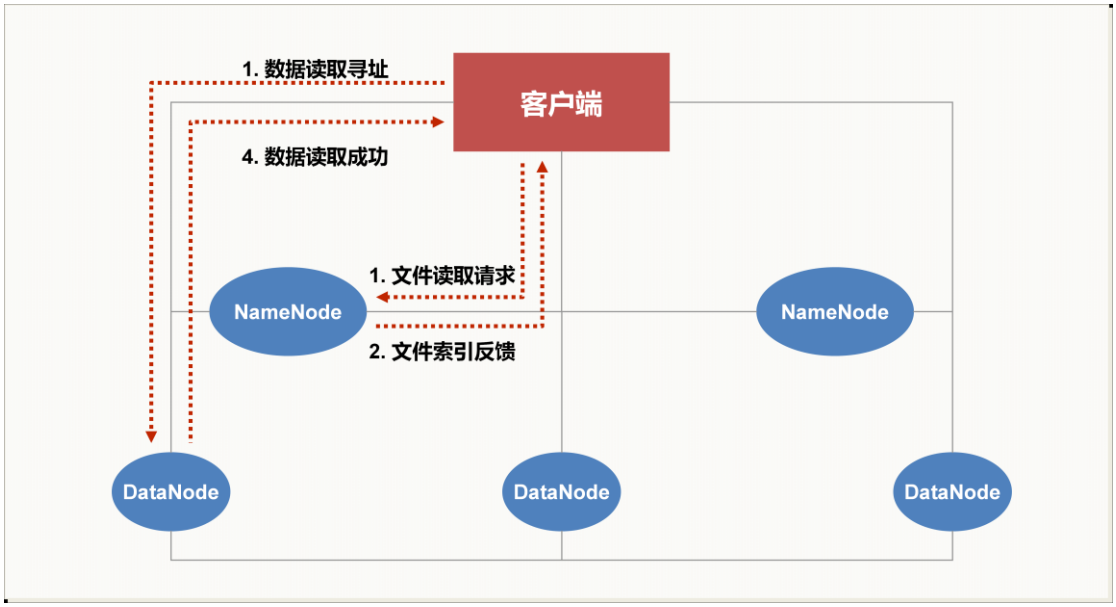
18、应用：小文件分布式文件系统 HsDFS

基于平台的针对海量小文件的分布式文件系统（HSDFS），满足当前电商、互联网分析与搜索、交通、科研等需求。Name Node 储存文件位置信息和文件目录结构信息，采用内存缓存热点信息，支持百亿级文件量。Data Node 存储文件内容，多个小文件组成一个 64M 数据块，Data Node 内存缓存热点数据块。每个小文件支持 3 个备份。HSDFS 支持 PB 级存储量。在普通 7200 转 SATA 机械磁盘上日吞吐量超过 4TB。



图例说明：

根据实时负载状况，文件内容流式写入各个 Data Node，文件索引信息同时写入互为备份的两个 Name Node 中。部分成功的流程触发日志写入操作，以备后期完备化。



图例说明：

根据实时负载状况，从负载较轻的 Name Node 提取文件索引信息，根据索引信息和实时负载状况，从负载较轻的 Data Node 读取文件内容。

HsDFS 创建过程如下：

假设 HsDFS 尚未创建，依次启动 NameNode 和 DataNode，然后启动调测口。

- 创建 NameNode 的命令格式为

```
hsdfs create npp <root dir> mode <4K|64K|1M|2M|128M|256M|512M|1G|2G|4G> max
```

```
<num> disk max <num> np on tcid <tcid> at <console|log>
```

参数意义如下：

Root dir：表示 NameNode 持久化相关信息的存放目录

Mode：表示 NameNode 块的大小

Max：表示 NameNode 总的块数

Disk max：表示 NameNode 持久化信息存放的磁盘数，在存放目录下表示为子目录 disk0,disk1,…

Tcid：表示该 NameNode 部署在此平台节点上

At：表示当前命令执行结果输出在屏幕上还是日志文件中

比如，

```
hsdfs create npp /home/hansoul/hsdfs/10.10.10.1 mode 4G max 2 disk max 4 np on tcid 10.10.10.1 at console
```

- 创建 DataNode 的命令格式为

```
hsdfs create dn <root dir> with <num> disk <max> GB on tcid <tcid> at <console|log>
```

参数意义如下：

Root dir：表示 DataNode 持久化相关信息的存放目录

num：表示 DataNode 持久化信息存放的磁盘数，在存放目录下表示为子目录 disk0,disk1,…

Max：表示每块磁盘用来存储 DataNode 数据的磁盘空间大小

Tcid：表示该 DataNode 部署在此平台节点上

At：表示当前命令执行结果输出在屏幕上还是日志文件中

创建一个 HsDFS 含 2 x NameNode + 3 x DataNode 的命令组举例：

```
hsdfs create npp /home/ezhocha/hsdfs/10.10.10.1 mode 4G max 2 disk max 4 np on tcid 10.10.10.1 at console
hsdfs create npp /home/ezhocha/hsdfs/10.10.20.1 mode 4G max 2 disk max 4 np on tcid 10.10.20.1 at console

hsdfs create dn /home/ezhocha/hsdfs/10.10.10.2 with 100 disk 400 GB on tcid 10.10.10.2 at console
hsdfs create dn /home/ezhocha/hsdfs/10.10.10.3 with 100 disk 400 GB on tcid 10.10.10.3 at console
hsdfs create dn /home/ezhocha/hsdfs/10.10.10.4 with 100 disk 400 GB on tcid 10.10.10.4 at console
```

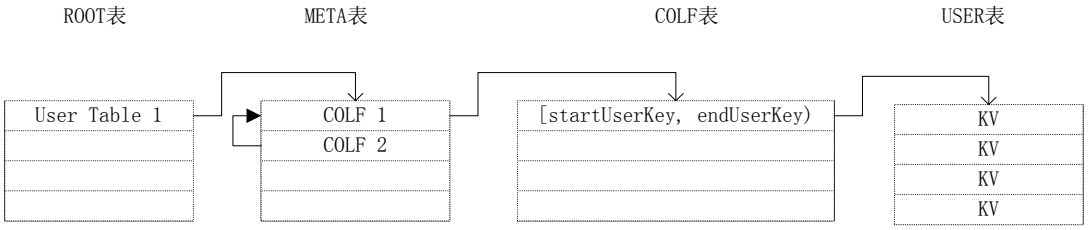
对 HsDFS 的访问方式，请参考 demo 以及调测口的 help 信息。

19、应用：NoSQL 分布式数据库 HsBGT

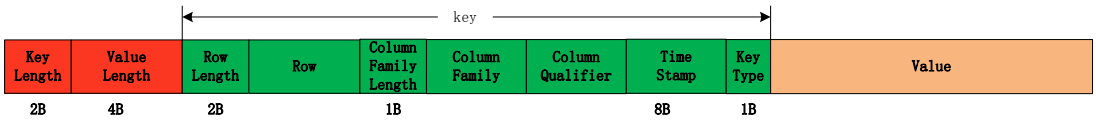
基于平台的针对海量非结构化数据存储的分布式数据库 HsBGT 以 KV 格式存储数据，用 B+树组织，支持百万级表服务器，其中表服务器分为 ROOT 表服务器、META 表服务器、COLF 表服务器和 USER 表服务器，支持表服务器的动态扩展。

【注】这里 USER 表服务器并非指用户定义的数据库表，而是存储用户数据的数据库表的子表，请避免混淆。

按目前的实现，ROOT 表仅支持一个，不可裂分。每个用户定义的数据库表对应唯一一个 META 表，不可裂分。一个 META 表管理多个 COLF 表，用来存储 COLUMN FAMILY 信息，COLF 表可裂分。一个 COLF 表管理多个 USER 表，用来存储 KV 信息，USER 表可裂分。

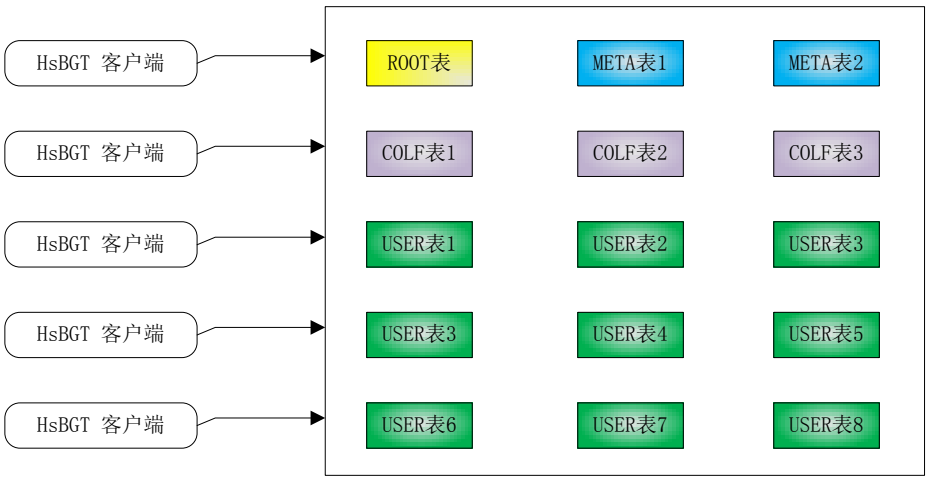


其中 KV 格式如下，



【注】与 HBASE 不同的是，平台设计者认为 Key Length 只有 2 个字节，即 Key 的长度最大支持到 64KB，已足够；而 HBASE 的 Key Length 为 4 个字节。另外，Key Type 在 HBASE 中用来表示一条 KV 记录是否被删除，平台设计者则是在删除一条 KV 记录时通过调整 B+树直接删除，因此事实上并未使用该标记。

HsBGT 表服务器集群：



其中 HsBGT 客户端会会在数据库访问过程，缓存一部分表服务器地址信息，以加速表服务器的定位和访问。

HsBGT 创建过程如下：

假设 HsBGT 尚未创建，依次启动 HsDFS 和 HsBGT 表服务器集群，然后启动调测口。

【注】HsBGT 的宿主分布式文件系统也可以是 NFS。

- 创建根表的命令格式为

```
hsbgt create root table <name> with path <path> on tcid <tcid> rank <rank> at <console|log>
```

参数意义如下：

Name：表示根表名称

path：表示根表持久化信息在 HsBGT 中的存放目录

Tcid：表示该根表部署在此平台节点上

rank：表示该根表部署在此进程上

At：表示当前命令执行结果输出在屏幕上还是日志文件中

比如，

```
hsbgt create root table root01 with path /home/ezhocha/hsbgt on tcid 10.10.10.1 rank 1 at console
```

- 创建用户表的命令格式为

```
hsbgt create user table <name> column family <col[:col]> from path <path> at <console|log>
```

参数意义如下：

Name：表示用户表名称

Column family：表示用户表的 column family 组

path：表示根表持久化信息在 HsBGT 中的存放目录

At：表示当前命令执行结果输出在屏幕上还是日志文件中

比如，

```
hsbgt create user table hansoul column family colf-0:colf-1:colf-2 from path /home/ezhocha/hsbgt at console
```

【注】用户表可以手动命令行方式创建，也可以编程动态创建。

对 HsBGT 的访问方式，请参考 demo 以及调测口的 help 信息。

20、应用：其它

大数软件包、有限域软件包、椭圆曲线及其密码系统等，详情请参见源码。

21、平台任务接口

任务接口负责任务创建，远端模块的启停，任务广（组）播，任务点播、任务点到点等功能。

21.1 平台动态负载均衡策略

负载均衡策略	编号	描述
LOAD_BALANCING_LOOP	0	轮询策略。轮流向各模块派发任务。
LOAD_BALANCING_MOD	1	模块负载策略。向负载最小的模块派发任务。
LOAD_BALANCING_RANK	2	进程负载策略。向负载最小的进程中的模块派发任务。

21.2 任务优先级定义

任务优先级	编号	描述
TASK_PRIO_NORMAL	0	正常优先级。同级遵循先进先出原则（FIFO）。
TASK_PRIO_HIGH	1	高优先级。同级遵循先进先出原则（FIFO）。
TASK_PRIO_PREEMPT	2	抢占优先级。同级遵循后进先出原则（LIFO）。

优先级从低到高依次为 TASK_PRIO_NORMAL，TASK_PRIO_HIGH，TASK_PRIO_PREEMPT。优先级高的任务先于优先级低的任务被调度。

21.3 任务响应标志

任务响应标志	编号	描述
TASK_NEED_RSP_FLAG	1	任务发起者需要任务响应。
TASK_NOT_NEED_RSP_FLAG	2	任务发起者不需要任务响应。

21.4 任务重新调度标志

任务重新调度标志	编号	描述
TASK_NEED_RESCHEDULE_FLAG	3	远端模块断连时，重新调度任务请求到可达的远端模块。
TASK_NOT_NEED_RESCHEDULE_FLAG	4	远端模块断连时，不重新调度任务请求

21.5 任务响应收集标志

任务响应收集标志	编号	描述
TASK_NEED_ALL_RSP	-1	任务收集齐全部任务响应视为任务结束。
TASK_NEED_NONE_RSP	0	任务不需要收集任何任务响应。

当任务收集指定数量的任务响应时，设置成除以上两值之外的值即可。

21.6 任务生命期标志

任务生命期标志	编号	描述
TASK_ALWAYS_LIVE	-1	任务生命期直至任务结束。

注意：目前版本尚不支持任务生命期设定。

21.7 任务创建接口

任务创建接口包括创建任务模板和添加任务请求。

函数原型定义	void * task_new (const void *mod_mgr, const UINT32 task_prio, const UINT32 task_need_rsp_flag)			
函数功能描述	任务创建，创建一个任务管理者			
函数返回类型	void *			
函数返回描述	执行成功返回任务管理者指针，否则返回空指针			
参数序号	参数名称	参数类型	参数方向	参数描述
1	mod_mgr	void *	输入	模块管理者
2	task_prio	UINT32	输入	任务优先级
3	task_need_rsp_flag	UINT32	输入	任务响应标志
4	task_mgr_ret	void **	输出	返回任务管理者
5				

函数原型定义	UINT32 task_inc(void *task_mgr,const void * func_retval_addr, const UINT32 func_id, ...)			
函数功能描述	向任务管理者中添加一个函数调用的任务请求			
函数返回类型	int			
函数返回描述	执行成功返回 0			
参数序号	参数名称	参数类型	参数方向	参数描述
1	task_mgr	void *	输出	任务管理者
2	func_retval_addr	void *	输入	函数返回值地址
3	func_id	UINT32	输入	函数唯一索引
4				
5				

函数原型定义	UINT32 task_pos_inc(void *task_mgr, const UINT32 recv_mod_node_pos, const void * func_retval_addr, const UINT32 func_id, ...)			
函数功能描述	向任务管理者中指定的模块添加一个函数调用的任务请求			
函数返回类型	UINT32			
函数返回描述	执行成功返回 0；执行失败返回-1			
参数序号	参数名称	参数类型	参数方向	参数描述
1	task_mgr	void *	输出	任务管理者

2	recv_mod_node_pos	UINT32	输入	指定模块的位置
3	func_retval_addr	void *	输入	函数返回值地址
4	func_id	UINT32	输入	函数唯一索引
5				

函数原型定义	UINT32 task_tcid_inc(TASK_MGR *task_mgr, const UINT32 recv_tcid, const void * func_retval_addr, const UINT32 func_id, ...)			
函数功能描述	向任务管理者中指定的通信子中负载最小的模块添加一个函数调用的任务请求			
函数返回类型	int			
函数返回描述	执行成功返回 0			
参数序号	参数名称	参数类型	参数方向	参数描述
1	task_mgr	void *	输出	任务管理者
2	recv_tcid	UINT32	输入	指定通信子标识
3	func_retval_addr	void *	输入	函数返回值地址
4	func_id	UINT32	输入	函数唯一索引
5				

函数原型定义	UINT32 task_super_inc(TASK_MGR *task_mgr, const MOD_NODE *send_mod_node, const MOD_NODE *recv_mod_node, const void * func_retval_addr, const UINT32 func_id, ...)			
函数功能描述	指定发送模块向任务管理者中指定的接收模块添加一个函数调用的任务请求			
函数返回类型	int			
函数返回描述	执行成功返回 0			
参数序号	参数名称	参数类型	参数方向	参数描述
1	task_mgr	void *	输出	任务管理者
2	send_mod_node	MOD_NODE *	输入	指定发送模块
3	recv_mod_node	MOD_NODE *	输入	指定接收模块
4	func_retval_addr	void *	输入	函数返回值地址
5	func_id	UINT32	输入	函数唯一索引

21.8 远端模块激活接口

函数原型定义	UINT32 task_act (const void *src_mod_mgr, void **des_mod_mgr, const UINT32 mod_num, const UINT32 load_balancing_choice, const UINT32 task_prio, const UINT32 func_id, ...)

函数功能描述	启动远端模块组，同时生成模块管理者			
函数返回类型	UINT32			
函数返回描述	执行成功返回 0；执行失败返回-1			
参数序号	参数名称	参数类型	参 数 方向	参数描述
1	src_mod_mgr	void *	输入	源模块管理者
2	des_mod_mgr	void *	输出	目标模块管理者
3	mod_num	UINT32	输入	待启动模块数量
4	load_balancing_choice	UINT32	输入	动态负载均衡策略
5	task_prio	UINT32	输入	任务优先级
6	func_id	UINT32	输入	模块启动函数唯一索引

21.9 远端模块去激活接口

函数原型定义	UINT32 task_dea (void *mod_mgr, const UINT32 task_prio, const UINT32 func_id, ...)			
函数功能描述	停止远端模块组，同时释放模块管理者			
函数返回类型	UINT32			
函数返回描述	执行成功返回 0；执行失败返回-1			
参数序号	参数名称	参数类型	参数方向	参数描述
1	mod_mgr	void *	输出	模块管理者
2	task_prio	UINT32	输入	任务优先级
3	func_id	UINT32	输入	模块停止函数唯一索引
4				
5				

21.10任务广播接口

函数原型定义	UINT32 task_bcast(const void *mod_mgr, const UINT32 task_need_rsp_flag, const UINT32 task_prio, const UINT32 func_id, ...)			
函数功能描述	向指定模块管理者中的所有模块广播任务			
函数返回类型	UINT32			
函数返回描述	执行成功返回 0；执行失败返回-1			
参数序号	参数名称	参数类型	参数方向	参数描述
1	mod_mgr	void *	输入	模块管理者
2	task_need_rsp_flag	UINT32	输入	任务响应标志
3	task_prio	UINT32	输入	任务优先级
4	func_id	UINT32	输入	函数唯一索引
5				

21.11任务点播接口

函数原型定义	UINT32 task_mono(const void*mod_mgr, const UINT32 task_need_rsp_flag, const UINT32 task_prio, const void * func_retval_addr, const UINT32 func_id, ...)			
函数功能描述	向指定模块管理者中的添加点播任务，目标执行模块有平台动态负载平衡决定			
函数返回类型	UINT32			
函数返回描述	执行成功返回 0；执行失败返回-1			
参数序号	参数名称	参数类型	参数方向	参数描述
1	mod_mgr	void *	输入	模块管理者
2	task_need_rsp_flag	UINT32	输入	任务响应标志
3	task_prio	UINT32	输入	任务优先级
4	func_retval_addr	void *	输入	函数返回值地址
5	func_id	UINT32	输入	函数唯一索引

函数原型定义	UINT32 task_pos_mono(const void *mod_mgr, const UINT32 task_need_rsp_flag, const UINT32 task_prio, const UINT32 recv_mod_node_pos, const void * func_retval_addr, const UINT32 func_id, ...)			
函数功能描述	向指定模块管理者中指定模块添加点播任务			
函数返回类型	UINT32			
函数返回描述	执行成功返回 0；执行失败返回-1			
参数序号	参数名称	参数类型	参数方向	参数描述
1	mod_mgr	void *	输入	模块管理者
2	task_need_rsp_flag	UINT32	输入	任务响应标志
3	task_prio	UINT32	输入	任务优先级
4	recv_mod_node_pos	UINT32	输入	指定模块的位置
5	func_retval_addr	void *	输入	函数返回值地址
6	func_id	UINT32	输入	函数唯一索引

函数原型定义	UINT32 task_tcid_mono(const MOD_MGR *mod_mgr, const UINT32 task_need_rsp_flag, const UINT32 task_prio, const UINT32 recv_tcid, const void * func_retval_addr, const UINT32 func_id, ...)			
函数功能描述	向指定模块管理者中指定通信子的负载最小的模块添加点播任务			
函数返回类型	UINT32			
函数返回描述	执行成功返回 0；执行失败返回-1			
参数序号	参数名称	参数类型	参数方向	参数描述
1	mod_mgr	void *	输入	模块管理者
2	task_need_rsp_flag	UINT32	输入	任务响应标志

3	task_prio	UINT32	输入	任务优先级
4	recv_tcid	UINT32	输入	指定通信子标识
5	func_retval_addr	void *	输入	函数返回值地址
6	func_id	UINT32	输入	函数唯一索引

函数原型定义	UINT32 task_super_mono(const MOD_MGR *mod_mgr, const UINT32 task_need_rsp_flag, const UINT32 task_prio, const MOD_NODE *recv_mod_node, const void * func_retval_addr, const UINT32 func_id, ...)			
函数功能描述	向指定模块管理者中指定通信子的负载最小的模块添加点播任务			
函数返回类型	UINT32			
函数返回描述	执行成功返回 0；执行失败返回-1			
参数序号	参数名称	参数类型	参数方向	参数描述
1	mod_mgr	void *	输入	模块管理者
2	task_need_rsp_flag	UINT32	输入	任务响应标志
3	recv_mod_node	MOD_NODE *	输入	指定接收模块
4	func_retval_addr	void *	输入	函数返回值地址
5	func_id	UINT32	输入	函数唯一索引
6				

21.12 任务点到点接口

函数原型定义	UINT32 task_p2p(const UINT32 modi, const UINT32 time_to_live, const UINT32 task_prio, const UINT32 task_need_rsp_flag, const UINT32 task_need_rsp_num, const void *recv_mod_node, const void * func_retval_addr, const UINT32 func_id, ...)			
函数功能描述	向指定的远端模块发起一次任务			
函数返回类型	UINT32			
函数返回描述	执行成功返回 0；执行失败返回-1			
参数序号	参数名称	参数类型	参数方向	参数描述
1	modi	UINT32	输入	模块标识
2	time_to_live	UINT32	输入	任务生命周期。始终置为 TASK_ALWAYS_LIVE
3	task_prio	UINT32	输入	任务优先级
4	task_need_rsp_flag	UINT32	输入	任务响应标志
5	task_need_rsp_num	UINT32	输入	任务响应收集数量。如果任务需要收齐全部响应，则置为任务响应收集标志 TASK_NEED_ALL_RSP；如果任务无响应，则置为任

				务响应收集标志 TASK_NEED_NONE_RSP ；如果任务收集指定数量的响应，则置为指定数值。
6	recv_mod_node	void *	输入	远端模块
7	func_retval_addr	void *	输入	函数返回值地址
8	func_id	UINT32	输入	函数唯一索引

函数原型定义	UINT32 task_p2p_inc(void *task_mgr, const UINT32 modi, const void *recv_mod_node, const void * func_retval_addr, const UINT32 func_id, ...)			
函数功能描述	向任务管理者中添加一个点到点的任务请求			
函数返回类型	UINT32			
函数返回描述	执行成功返回 0；执行失败返回-1			
参数序号	参数名称	参数类型	参数方向	参数描述
1	task_mgr	void *	输入输出	模块管理者
2	modi	UINT32	输入	模块标识
3	recv_mod_node	void *	输入	远端模块
4	func_retval_addr	void *	输入	函数返回值地址
5	func_id	UINT32	输入	函数唯一索引

21.13任务阻塞接口

函数原型定义	EC_BOOL task_wait(TASK_MGR *task_mgr, const UINT32 time_to_live, const UINT32 task_need_rsp_num, const UINT32 task_reschedule_flag, CHECKER ret_val_checker)			
函数功能描述	向平台提交任务管理者的所有任务请求，并阻塞等待。如果任务有响应，则等到收齐指定数量响应后返回；如果任务无响应，则等到任务全部提交完毕后返回。用户应用负责任务管理者所关联的模块管理者的内存释放，但不用负责任务管理者的内存释放。			
函数返回类型	EC_BOOL			
函数返回描述	执行成功返回 EC_TRUE；执行失败返回 EC_FALSE			
参数序号	参数名称	参数类型	参数方向	参数描述
1	task_mgr	void *	输入	模块管理者
2	time_to_live	UINT32	输入	任务生命周期。始终置为 TASK_ALWAYS_LIVE
3	task_need_rsp_num	UINT32	输入	任务响应收集数量。如果任务需要收齐全部响应，则置

				为任务响应收集标志 TASK_NEED_ALL_RSP；如果任务无响应，则置为任务响应收集标志 TASK_NEED_NONE_RSP；如果任务收集指定数量的响应，则置为指定数值。
4	task_reschedule_flag	UINT32	输入	任务重新调度标志
5	ret_val_checker	CHECKER	输入	任务响应检查函数指针。检查通过通过，该函数返回 EC_TRUE，否则返回 EC_FALSE。一般情况下，可置该函数指针为 NULL_PTR。

函数原型定义	EC_BOOL task_no_wait(TASK_MGR *task_mgr, const UINT32 time_to_live, const UINT32 task_need_rsp_num, const UINT32 task_reschedule_flag, CHECKER ret_val_checker)			
函数功能描述	向平台提交任务管理者的所有任务请求后，立即返回。用户应用不负责任务管理者和模块管理者的内存释放。			
函数返回类型	EC_BOOL			
函数返回描述	执行成功返回 EC_TRUE；执行失败返回 EC_FALSE			
参数序号	参数名称	参数类型	参数方向	参数描述
1	task_mgr	void *	输入	模块管理者
2	time_to_live	UINT32	输入	任务生命周期。始终置为 TASK_ALWAYS_LIVE
3	task_need_rsp_num	UINT32	输入	任务响应收集数量。如果任务需要收齐全部响应，则置为任务响应收集标志 TASK_NEED_ALL_RSP；如果任务无响应，则置为任务响应收集标志 TASK_NEED_NONE_RSP；如果任务收集指定数量的响应，则置为指定数值。
4	task_reschedule_flag	UINT32	输入	任务重新调度标志
5	ret_val_checker	CHECKER	输入	任务响应检查函数指针。检查通过通过，该函数返回 EC_TRUE，否则返回 EC_FALSE。一般情况下，可置该函数指针为 NULL_PTR。

22、平台许可证

平台采用 GPLv2 开源许可证发布。按照 GPLv2 开源许可证，其内容包括：使用了 GPLv2 代码的软件在发布时必须使用 GPLv2 许可证发布源代码。

为了便于推广使用，平台许可证设置了一些例外，即在 GPLv2 许可证之外可以向使用者发布商业应用的商业许可证。目前平台全部代码均为作者周超勇编写。

当用户遵循以下两个前提条件时：
1. 在平台中保留 LOGO
2. 在产品说明中声明使用本平台

经审核后发放相应的商业许可证。这个申请流程是：
1、用户提出申请，并填写下面的申请表格给出相应的一些信息
2、审核后给出商业许可证协议的电子文本
3、用户对电子文本无异议后，打印商业许可协议并盖章，然后快递给我们
4、我们收到后，盖章或签字后再寄回给企业。
这其中寄送的快递费用由用户承担。免费商业许可证申请表格如下：

公司名称	
公司地址	
联 系 人	
联系电话	
申请条件	☐ 保留分布式计算平台 LOGO ☐ 在产品使用说明书中声明使用分布式计算平台
产品简介	
（包含：产品简要介绍、应用领域、使用到的分布式计算平台功能情况）	
产品图片	
（申请免费商业许可证，此项是必须的）	

表格信息填写完毕后，需要邮件发送到：bgnvendor@gmail.com。通常我们收到申请后会给出回复，如果没有收到回复，有可能被 Google Mail 当作垃圾邮件处理了。遇到这种情况，请多发送几次。
对于不满足前面提到的条件的企业用户，或希望申请纯商业许可证的企业用户（例如考虑到

商业保密，技术保障支持服务等），也可以向我们购买纯商业使用授权【注 2】。

【注 1】

- 对于企业已经开始销售的产品，我们有权不发放商业许可证，而是采用 GPLv2 许可证处理；
- 对于未有商业许可证且未按照 GPLv2 许可证发布代码的用户，我们将联合国内开源届的基金协会、律师向企业提起诉讼以维护我们的权利；

【注 2】可以向 bgnvendor@gmail.com 联系纯商业使用授权，并在邮件文本上进行注明。