YarnPlus：基于 Yarn 的异构任务资源共享框架

王文峰，胡春明，沃天宇，杨任宇*
（北京航空航天大学计算机学院，北京，100191）
Email: {wangwf, hucm, woty, yangry}@act.buaa.edu.cn

摘　　要：在数据中心和运行大规模数据处理计算的云环境中有效的资源管理不仅可以提高作业的运行性能还能够提高集群资源利用率。随着不同计算框架与相应负载的出现，高效地资源共享方案可以使得不同负载任务共享同一集群来提升集群的利用率和效率。针对不同任务共享集群的需求，本文从 Hadoop MapReduce job 与虚拟机两类典型的数据中心任务出发，设计了多任务共享资源的解决方案，利用新一代 Hadoop Mapreduce 框架 Yarn 资源分派与任务调度解耦和的特性，对其扩展支持虚拟机生命周期的管理，扩展后的 Yarn 框架 YarnPlus 可以根据全局资源的使用与请求情况在不同任务间动态分派资源，避免了资源冲突与竞争，保证了任务的资源需求。实验表明扩展后的 Yarn 系统 YarnPlus 在管理虚拟机和 Hadoop Job 两类任务时，不仅能保证 VM 服务的性能，同时跟未统一考虑全局资源与任务分派情况相比，缩短了 MapReduce job 的执行时间，提升了数据处理效率。最终通过 YarnPlus 实现了异构任务间的细粒度资源共享，减少了集群所需的硬件资源，节省了成本。

关　键　词：资源共享；异构任务；性能保证；Hadoop; Yarn

YarnPlus: A Yarn-based Framework for Fine-Grained Resource Sharing in Heterogeneous Job

Wenfeng Wang, Chunming Hu, Tianyu Wo, Renyu Yang*
School of Computing, Beihang University, Beijing, China
Email: {wangwf, hucm, woty, yangry}@act.buaa.edu.cn

Abstract: Efficient resource management in data centers and clouds running large data processing frameworks like Hadoop is crucial for the performance enhancement of hosted applications and the improvement of resource utilization. As the emerging of new computing frameworks and their workloads, it is significantly important to share fine-granined resources among multiple diverse workloads in one specific data center. For this reason, we extend YARN, the next generation of map reduce computing framework, in order to support the management of lifecycle of virtual machine while running offline computing tasks. In addition, we develop a platform called YarnPlus for fine-grained resource sharing among MapReduce jobs and virtual machine services. The experimental results illustrate that YarnPlus has a better performance in managing cluster resource and running heterogeneous jobs due to the fact that it takes cluster resource utilization as well as heterogeneous job resource requests into account when dynamic resource allocation. Finally, YarnPlus makes fine-grained resource sharing in heterogeneous job available and can reduce the amount of hardware required for a workload to save the cost.

Key words: Resource sharing; Heterogeneous job; Performance guarantee; Hadoop; Yarn

引言[1]

随着网络服务数量与规模的增大，越来越多的服务在云计算环境中运行。这些服务的提供需要大规模数据处理支持，同时需要对产生的数据进行分析处理从而更改进服务的质量。MapReduce[1]已成为云环境中一种重要的大规模数据处理编程模型，包括 Facebook 和 Yahoo！在内的许多公司与学术机构都采用了 MapReduce 的开源实现 Apache Hadoop[2]。这样，数据处理与分析的需求使得互联网公司、研究机构及其他企业部署了用来处理大规模 MapReduce 任务的 Hadoop 集群。

同时随着网络带宽不断增长，通过网络访问非

收稿日期： 2013-06-20
作者简介：王文峰，男，硕士研究生，研究方向为云计算、任务调度；胡春明，男，博士，副教授，研究方向为云计算，虚拟化；沃天宇，男，博士，副教授，研究方向为分布式计算，虚拟化，资源管理等；通信作者*：杨任宇，男，博士研究生，研究方向为虚拟计算环境、分布式计算。
基金项目：以支撑电子商务为主的网络操作系统研制（2011AA01A202）

本地化的计算服务（包括应用、存储和信息服务等）的条件越来越成熟，其中由于虚拟化技术取得的突破使得 IaaS (Infrastructure as a Service)发展尤其迅速，学术界与工业界都涌现出虚拟化云计算平台提供虚拟机服务。

不同需求与研究使得工业界及学术界对不同任务和服务有不同的计算集群，然而为了满足峰值资源的需求，各个集群都是采用最大资源量来部署，这样使得集群中资源利用率较低，而且不同业务集群的资源需求不同，使得资源的空闲浪费尤其严重。Facebook 2000 节点的 Hadoop 集群的分析[16]表明集群资源利用率不高，其中内存平均利用率不到 30%。

集群资源利用率不高，主要是由于不同任务独占集群引起的空闲资源浪费，通过合理调度使得不同任务共享集群可以减少浪费提高资源利用率。然而现有的单层调度器将资源分派与作业调度由同一进程负责的特性不适用于异构任务的调度。双层调度器由元调度器负责将资源分派给任务相关的子调度器，而子调度器负责作业的调度与监控，这样不仅支持了多任务的调度，而且增强了系统扩展性。然而双层资源调度平台 Mesos 在资源分派中采用的 Resource Offer 机制容易对资源需求高的大任务产生饥饿，同时资源撤销机制又不能保证长任务的执行性能，这样使得 Mesos 无法适用于单任务资源需求大、任务生命周期长的任务。本文从云环境中最典型的两类任务：Hadoop MapReduce job 与虚拟机 VM 任务出发，设计针对这两种任务可行的资源共享解决方案，并对不同方案从可行性到性能进行实验验证。根据结果，提出了一种基于 Yarn（the next generation hadoop mapreduce framework）的 VM 与 Hadoop MapReduce Job 间高效可靠地资源共享框架 YarnPlus，该框架能够根据全局资源使用情况和任务资源请求在异构任务间动态分派资源，避免了不同任务对相同资源竞争而产生的性能下降。从而在保证任务性能的前提下，实现了异构任务间的资源共享，减少了集群所需硬件资源，节省了成本。

本文主要贡献如下：

- 分析 MapReduce 任务与 VM 任务的不同特性、分析一些学术界研究的统一调度框架，在这基础上提出 VM 与 MapReduce job 可能的共享资源统一调度方案。
- 利用 Hadoop Yarn（Next Generation Apache Hadoop Mapreduce）资源分派与任务调度监控的解耦和，扩展 Yarn 使其支持 VM 服务，实现 VM 整个生命周期的管理，从而利用 Yarn 来完成 VM 与 MapReduce Job 的共享资源统一调度。
- 基于 hadoop-2.0.3-alpha 源代码实现系统，并从任务执行性能入手设计实验，分析几种可能的共享调度方案。结果表明扩展后的 YarnPlus 在 VM 与 MapReduce Job 间共享资源可以很好的保证任务的性能。

本文的后续部分安排如下：第 1 节阐述一些背景和相关工作；第 2 节通过分析 Yarn 的结构，对 Yarn 进行扩展的结构设计与功能实现，提出可行的资源共享调度方案 YarnPlus。第 3 节描述性能实验结果并进行讨论，同时提出可改进的问题；第 4 节是结论和今后的工作介绍。

1 背景与相关工作

随着不同计算框架与服务的出现，每种类型的作业或者服务独占一个集群，而集群的峰值部署和任务资源使用偏好使得集群资源利用率较低。一种可能的解决方案是通过整合不同任务负载共享集群，高效合理地使用集群资源进而减少集群中空闲资源浪费从而提升集群资源利用率。这些不同的负载可以是：CPU 密集型任务和 Memory 密集型任务；小任务和大任务以及批处理任务（以 Hadoop MapReduce 为代表）和低延迟服务（以 VM 及存储服务为代表）。然而不同任务的负载特性，使得关于任务与资源的调度决策更加复杂，简单的任务混合不能考虑其他任务对资源的使用，容易产生资源竞争而影响任务的执行性能。因此，需要一种可以考虑整体资源视图，根据不同任务资源需求在异构任务间动态分派资源的调度框架来完成异构任务间的资源共享。

本节首先分析 MapReduce Job 与 VM 两类任务的异构性，然后介绍国内外现有的多任务资源共享研究，并分析其在支持 MapReduce Job 与 VM 共享资源的扩展性。

1.1 任务异构性

Hadoop MapReduce Job 与虚拟机 VM 是两类不同的任务。MapReduce Job 是一类关心任务吞吐量的离线批处理任务，而 VM 是一类更在乎性能并且对延迟敏感的服务类任务。通过 Facebook Hadoop 集群中 MapReduce job 和 task 的执行时间的累积分布函数（CDF）。可以看出 MapReduce job 的任务持续时间都比较短。

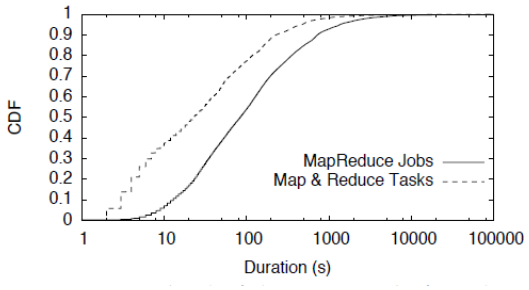

图 1 CDF of job and task durations in facebook’s hadoop data warehouse[10]

同时 Amazon 提供的虚拟机 VM 服务 EC2 的计费最小时间为 1h（3600s），超过 40%的的服务持续时间超过 1 个月，即 VM 是一类长任务。MapReduce job 作为一种离线短任务失效重启代价不大，所以一个轻量级的简单的任务分派方法即可以满足需求。但是对于 VM 这样提供在线服务的长任务，必须对其保证严格的资源可用性和性能指标，即必须仔细合理地分派任务，保证 VM 服务有高性能和低失效。

1.2 多任务资源共享方案研究

关于多任务资源共享的调度现主要有单层的中央调度框架和双层的并行调度框架。

单层调度框架的特点是所有的资源调度和作业管理的功能全部放到同一个进程去完成。这种方式在高性能计算 HPC 中比较常见，HPC 调度器通常是一个独立的进程，对所有任务应用相同的算法，且调度器必须明确各个任务详细的调度及任务分解流程。虽然以 Maui 和 Moab 为代表的一些 HPC 调度器通过对所有任务优先级进行多权重因子计算后可以支持不同的调度策略。但是这种中央调度器在多任务统一调度中面临着一些不足：首先，如果要支持不同类型的任务，则调度器中必须实现各种任务原有的调度结构，最终会导致单层调度器实现上的臃肿与更新上的复杂；其次，对于当原有调度系统需要扩展新任务时，对现有调度器的改动非常大，很容易影响到已经支持任务的调度；最后，这种调度结构使得调度器很容易成为系统的瓶颈所在，复杂的调度结构不仅增加了资源到任务的决策时间，而且当调度器单点失效会造成整个系统的失效。所以 HPC 调度器通常用在任务执行与分解逻辑相同的同构多任务调度中，无法支持 VM 与 MapReduce Job 这两类完全异构的任务。

双层并行调度框架分为静态划分和动态划分两种。静态双层调度架构是指将整个集群资源划分为不同部分交给不同类型任务使用，允许不同任务各自的调度器根据自己获得的集群资源并行做出决策。主要是因为云计算中的调度器总是在自己拥有的一部分集群资源上做出关于任务的调度并监控任务的运行，通过将集群资源中合理的子集交给 Hadoop 子调度器可以最少程度修改 Hadoop。然而这种方法会导致产生集群中产生碎片，而且当不同任务队资源需求发生变化时，不能在负载间根据需求变化划分集群资源。动态双层调度比起静态划分，最主要的功能就是通过一个中央调度器根据负载情况将集群资源动态划分给子调度器使用。这种双层调度框架以 Apache Mesos[12]为代表。

在 Mesos 中，一个中央资源分派器动态地划分集群，并将资源分派给不同的子调度框架，由各自的子调度框架根据获取到的资源对其任务进行调度，并监控任务的执行。资源分派主要是采用 Resource Offer 的形式，即将可用资源依次提供给不同的调度框架，其中资源供给量及顺序主要是通过由 dominant resource fairness（DRF）[16]决定。由于同一时刻只有一个调度器可以决定是否接受该资源供给，这样就是实现了一种锁机制保证某种资源在某一时期不会被不同子调度框架竞争。如下是 Apache Mesos 的体系结构图。

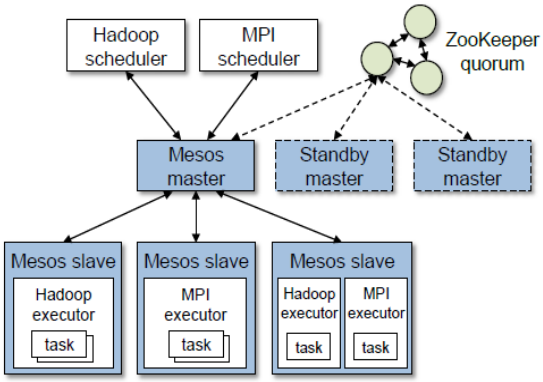

图 2 mesos 体系结构图

尽管 Mesos 现在已经支持了 Hadoop 和 MPI 这两类任务的计算，而且还可以将其扩展支持其他任务。但是 Mesos 依然面临着一些缺点：首先，Mesos 只有在任务生命周期较短并且所需资源量较小的情况下执行效果最好。这主要是由于 Mesos 采用的 resource offer 机制，当有空闲资源时，小任务首先得到满足，而大任务容易产生饥饿；其次，为了保证某种资源不会贪婪地被子调度器长时间占有，Mesos 会撤销长时间分派的资源，强制任务结束。这个特性使得 Mesos 对于 VM 这种生命周期长的任务不能够提供性能保证；最后，Mesos 采用的

resource offer 机制对于 MapReduce job 很难满足其数据本地性，因为 Mesos 提供的可用资源并不是拥有数据本地性的资源。

通过以上分析，可以发现现有的多任务统一调度方案不仅很难直接应用在 VM 与 Hadoop job 上实现资源共享，而且任务执行的性能不能保证。

2 设计与实现

本节首先结合已有系统和方法的借鉴之处和不足，提出针对这两种任务的资源共享方案，接着分析 Yarn 的特性并对 Yarn 进行结构扩展与功能实现，使其可以完成 VM 的完整生命周期的管理，从而实现 VM 与 Hadoop job 的资源共享。

为解决现有系统的上述问题，系统基于 hadoop-2.0.3-alpha 源代码完成。

2.1 简单混合共享方案

基于对静态划分双层调度的分析，一些简单的策略往往会带来不错的效果，因此设想在集群中同时部署 Hadoop 系统和 IVIC 系统（一种基于 KVM 的虚拟化资源管理系统[21]），两个系统共享地使用整个集群。IVIC 系统调度器将所有虚拟机负载均衡的分派到各个节点中，从而保证各个节点长任务（VM）占用资源相对一致，避免某些节点资源消耗过多而引起 Hadoop job 长尾问题。而 Hadoop 中各个节点配置相同的 slot，充分使用节点中剩余资源。这样就可能使集群的负载比较均衡的情况下不同类型任务共享集群资源。

2.2 基于 VM 统一资源共享方案

虚拟化技术，尤其是硬件虚拟化技术的发展，使得一台虚拟机拥有相同配置物理机完整的功能，且系统性能相差不大。因此可以设想利用虚拟化技术创建较高配置虚拟机，在这些虚拟机中部署 Hadoop 来运行 MapReduce 任务，然后对这些 Hadoop 虚拟机与普通虚拟机同时管理，从而不仅可以基于 VM 实现多任务资源共享，而且虚拟机技术使得 Hadoop 集群具有很好的扩展型和部署的便捷性。这样即实现不同任务共享集群资源，而且增加了集群管理的便捷性。

2.3 基于 Yarn 的资源共享方案

Yarn 是下一代 Apache MapReduce 框架。它的基本设计思想是将 Hadoop JobTracker 拆分为两个独立的服务：一个全局的资源管理器 Resource Manager（RM）和每个应用程序特有的 Application Master（App Mast）。其中 RM 负责整个系统资源的管理和分配，而 App Mast 则负责单个应用程序的管理。Yarn 已经内置了 MapReduce job 的 App Mast 用来支持 MapReduce 任务的运行及管理。根据 VM 任务运行特点，编写相应的 VM App Mast 并将其注册到 RM 中对 Yarn 进行扩展可以使得 Yarn 支持 VM 服务。当用户提交 MapReduce job 或 VM 任务后，由 RM 启动相应的 App Mast 负责管理执行相应任务，而 RM 根据全局资源的使用与请求情况在 App Mast 间分配资源，从而实现异构任务间的资源共享。

2.3.1Yarn 的结构

随着集群规模的扩大和负载数增加，导致 Hadoop JobTracker 在内存消耗，线程模型和扩展性/可靠性方便暴露出了缺点，因此 Apache 重写了 Hadoop，即现在的 Yarn（Next Generation Apache Hadoop Mapreduce）。Yarn 最基本的思想，是将原有 Hadoop JobTracker 所负责的资源管理与任务调度/监控分为两个进程来完成。即将原有 Hadoop 的调度结构扩展为双层调度结构，在该解决方案中包括两个重要模块：全局的资源管理器 Resource Manager（RM）和每个应用相关的任务资源管理器 Application Master（App Mast）。RM 负责资源管理将系统的资源分派到各个 AM，而由 AM 负责相应任务的调度并监控任务的执行。

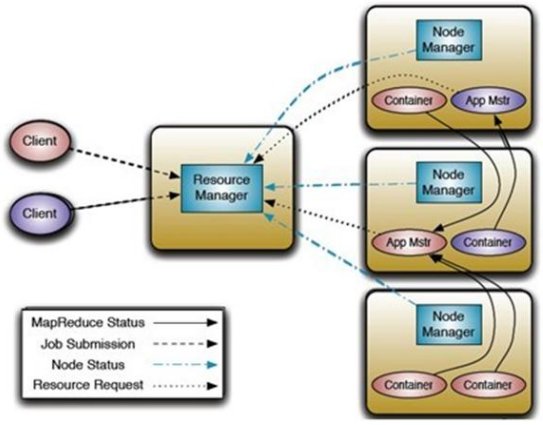

图 3 Yarn 体系结构

Yarn 总体上仍然是 master/slave 结构。在整个资源管理框架中，ResourceManager 为 master，NodeManager 为 slave，ResourceManager 负责对各个 NodeManager 上的资源进行统一管理和调度。

其中 Yarn 对资源的分派是由 Resource Manager(RM)和 Application Master(App Mast)相互协商来完成。APP Mast 不断向 RM 请求需要的资源量以及偏好节点（这样可以更好的实现数据本地性及应用相关偏好），RM 根据全局资源视图分派资源，当响应 App Mast 请求时，会向其返回封装了相应资源的 Container 容器列表，该列表包含了 container ID，包含的资源以及所在节点等。App

Mast 收到该响应则可与 Node Manage（NM）通信使用由 Container 封装的资源，其中 Yarn 采用 Linux Container 技术和 OS 线程监控技术实现 Container 中资源的隔离。

2.3.2 扩展后 YarnPlus 结构及工作流程

Yarn 中 Resource Manager 主要包括两部分，一部分是调度器 Scheduler 根据全局资源视图对 App Mast 的请求作出响应，另一部分是应用管理器 Applications Manager 用来管理 Yarn 中现有的 App Mast，并支持加入或撤销相应 App Mast。即 Yarn 允许用户通过编写相应的 Application client 和 Application Master 来扩展使用 Yarn 中资源。因此本文通过在 Yarn 中增加 VM 的 Application Clent 与 Application Master，从而扩展 Yarn 支持 VM 这种任务的执行和生命周期的管理，并最终通过 Yarn 来统一管理 MapReduce job 和 VM 共享集群资源。

让一种新的应用程序，运行于 Yarn 之上，用户需要编写两个组件完成该任务：客户端 Client 和 Application Master，其中，客户端负责向 ResourceManager 提交 ApplicationMaster，并查询应用程序运行状态，ApplicationMaster 负责向 ResourceManager 申请资源（以 Container 形式表示），并与 NodeManager 通信以启动各个 Container，此外，ApplicationMaster 还负责监控各个任务运行状态，并在任务失败时为其重新申请资源。MapReduce 作为一种通用的计算框架，Yarn 已经为其实现了一个直接可以使用的客户端—MRClient 和 ApplicationMaster—MRAppMaster，所以在 Yarn 中可以无需修改而运行以前版本 Hadoop MapReduce job。

在这两个组件编写中涉及到的 RPC 通信协议有：

1) ClientRMProtocol(Client<->ResourceManager)，Client 通过该协议将应用程序提交到 ResourceManager 上、查询应用程序的运行状态或者杀死应用程序等。
2) AMRMProtocol(ApplicationMaster<->ResourceManager)，ApplicationMaster 使用该协议向 ResourceManager 注册、申请资源以运行自己的各个任务。
3) ContainerManager(ApplicationMaster<->NodeManager)，ApplicationMaster 使用该协议要求 NodeManager 启动/撤销 Container，或者获取各个 container 的运行状态。

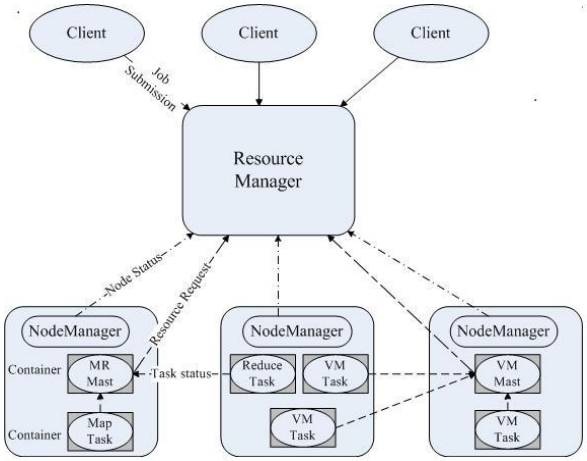

图 4 扩展后 YarnPlus 体系结构

扩展后的 YarnPlus 体系结构如图所示，主要由 ResourceManager、NodeManage、ApplicationMaster（图中给出了 MapReduce 和 VM 两种任务的 ApplicationMaster，分别为 MR Mast 和 VM Mast）和 Container 等几个组件构成。通过在 Yarn 中增加 VM Mast 来支持 VM 任务的运行与管理。其中 MR Mast 管理 MapReduce 任务，VM Mast 管理 VM 任务。各自的 Mast 负责根据任务资源需求向 RM 以容器的形式请求资源，然后进行资源任务绑定，调度任务到指定节点的容器中运行，同时负责监控任务的执行情况。

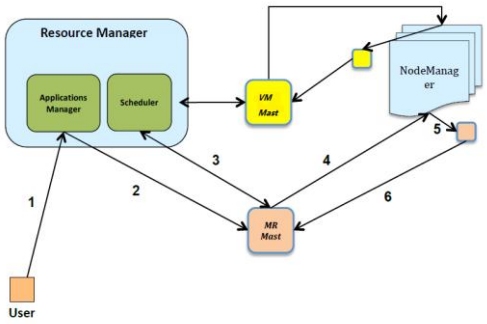

图 5 扩展后 YarnPlus 工作流程

扩展后 YarnPlus 具体工作流程为：1，用户提交作业；2，RM 根据用户提交作业类型，启动相应作业的 App Mast；3 该 App Mast（VM Mast 或者 MR Mast ）与 RM 的 scheduler 通信，不断请求所需资源；4，当获得请求资源后 App Mast 做出资源与任务的绑定，即根据相应资源调度任务；5， App Mast 与 NM 通信并获得由 RM 分派 Container 使用权，并在该 Container 中启动任务执行；6，任务运行后 Container 会定期汇报任务的执行情况到 App Mast 中供用户查看。通过以上的工作流程，RM 根据集群资源全局视图完成资源到 App Mast 的分派，

然后由相应 App Mast 根据获得资源调度相应类型任务的执行，这样便完成了不同任务间的资源共享，而且 Yarn 中自带的隔离机制可以减少任务间性能干扰。

2.3.3 Yarn 扩展的功能实现

2.3.3.1 VM 客户端 Client 的编写

客户端通常只需与 ResourceManager 交互，具体实现流程如下：

步骤 1：获取 Application Id。客户端通过 RPC 协议 ClientRMProtocol 向 ResourceManager 发送应用程序提交请求 GetNewApplicationRequest，ResourceManager 为其返回应答 GetNewApplicationResponse，该数据结构中包含多种信息，包括 ApplicationId、可用资源使用上限和下限等。

步骤 2： 提交 ApplicationMaster。客户端首先将启动 ApplicationMaster 所需的所有信息打包到数据结构 ApplicationSubmissionContext 中，主要包括以下几种信息：(1) Application Id；(2) Application 名称，(3) Application 优先级； (4) Application 所属队列； (5) Application 启动用户名；(6) ApplicationMaster 对应的 Container 信息，包括：启动 ApplicationMaster 所需各种文件资源、jar 包、环境变量、启动命令、运行 ApplicationMaster 所需的资源（主要指内存大小）等。

当封装好所需要传递的信息后，客户端会调用 ClientRMPProtocol#submitApplication(ApplicationSubmissionContext) 将 ApplicationMaster 提交到 ResourceManager 上。（ResourceManager 收到请求后，会为 ApplicationMaster 寻找合适的节点，并在该节点上启动它）。

2.3.3.2 VM ApplicationMaster 的编写

Application Master 需要与 ResoureManager 和 NodeManager 交互，以申请资源和启动 Container。具体步骤如下：

步骤 1，注册。ApplicationMaster 首先需通过 RPC 协议 AMRMProtocol 向 ResourceManager 发送注册请求 RegisterApplicationMasterRequest，该数据结构中包含 ApplicationMaster 所在节点的 host、RPC port 和 TrackingUrl 等信息，而 ResourceManager 将返回 RegisterApplicationMasterResponse，该数据结构中包含该应用程序的 ACL 列表、资源使用上限和下限等。

步骤 2，申请资源。根据每个任务的资源需求，Application Master 可向 ResourceManager 申请一系列用于运行任务的 Container，ApplicationMaster 使用 ResourceRequest 类描述每个 Container（一个 container 只能运行一个任务）：

表 1 ResourceRequest 类主要成员

成员变量	解释
Hostname	期望 Container 所在的节点，如果是“*”，表示可以为任意节点
Resource capability	运行该任务所需的资源量，当前仅支持内存资源
Priority	一个应用程序中的任务可能有多种优先级，ResourceManager 会优先为高优先级的任务分配资源
numContainers	符合以上条件的 container 数目

一旦为任务构造了 Container 后，ApplicationMaster 会使用 RPC 函数 AMRMProtocol#allocate 向 ResourceManager 发送一个 AllocateRequest 对象，以请求分配这些 Container。

表 2 AllocateRequest 类主要成员

成员变量	解释
Requested Containers	ResourceRequest 类对象，所需的 Container 列表
Released containers	ResourceRequest 类对象，由于失效需要释放的 Container 列表
Progress update information	应用程序执行进度
ResponseId RPC	响应 ID，每次调用 RPC，该值会加 1

ResourceManager 会为 ApplicationMaster 返回一个 AllocateResponse 对象，该对象中主要信息包含在 AMResponse 中：

表 3 AllocateResponse 类主要成员

成员变量	解释
Reboot	ApplicationMaster 是否需要重新初始化.
Allocated Containers	ResourceRequest 类对象，新分配的 container 列表
Completed Containers	已运行完成的 container 列表，该列表中包含运行成功和未成功的 Container

ApplicationMaster 会不断追踪已经获取的 container，且只有当需求发生变化时，才允许重新为 Container 申请资源。

步骤 3，启动 Container。当 ApplicationMaster（从 ResourceManager 端）收到新分配的 Container 列表后，会使用 RPC 函数 ContainerManager#startContainer 向对应的 NodeManager 发送 ContainerLaunchContext 以启动 Container。

表 4 ContainerLaunchContext 类主要成员

成员变量	解释
ContainerId	Container id.
Resource	该 Container 可使用的资源量（当前仅支持内存）
User	Container 所属用户
Security tokens	安全令牌，只有持有该令牌才可启动 container
LocalResource	运行 Container 所需的本地资源，如 jar 包、二进制文件、其他外部文件等
ServiceData	应用程序可能使用其他外部服务，这些服务相关的数据通过该参数指定
Environment	启动 container 所需的环境变量
command	启动 container 的命令

ApplicationMaster 会不断重复步骤 2~3，直到所有任务运行成功，此时，它会调用 AMRMProtocol#finishApplicationMaster，以告诉 ResourceManage 自己运行结束。

将对虚拟机的生命周期的管理与资源分派功能扩展到 Yarn 以后实现 YarnPlus，YarnPlus 可以实现机 VM 与 MapReduce job 统一管理，即可以根据全局资源使用与分派情况让两种异构任务更合理地共享集群资源，从而提升任务执行效率并提高集群资源利用率。

3 实验与讨论

本节首先介绍实验环境，然后针对不同的资源共享方案设计功能和性能实验，并实施实验分析实验结果，从而验证可行的资源共享解决方案并进行性能分析。下表是实验环境：

表 5 物理机硬件/软件环境

硬件/软件	参数
CPU	Intel(R) Core(TM) i7 2.80GHz
Memory	16G
操作系统	Debian 6.0 AMD 64
Hadoop	hadoop 1.0.4
Yarn	修改后的 hadoop-2.0.3-alpha
kvm	qemu-kvm-0.12.5

3.1 功能测试

本节搭建简单混合系统（hadoop merge vm）与对 Yarn 扩展后的 YarnPlus 系统验证系统中异构任务的正常运行。

首先选取 4 台上述节点，搭建集群 A，及简单混合系统。在其上分别搭建 Hadoop1.0.4 和 IVIC3.0（一套封装过提供 kvm 虚拟资源管理的系统，可以提供 VM 服务等）。搭建成功后分别启动 Hadoop 系统和 IVIC 系统，然后运行 hadoop-example 执行 WordCount，job 正常运行结束。在作业运行期间利用 IVIC 启动虚拟机。利用 VNC 连接虚拟机可正常使用。

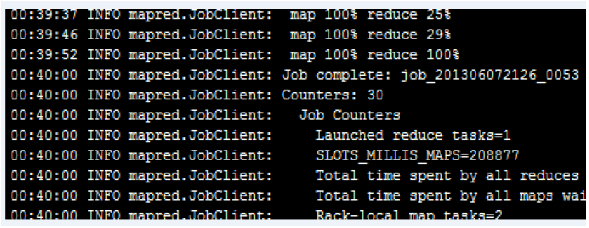

图 6 集群 A 中执行 MapReduce 作业

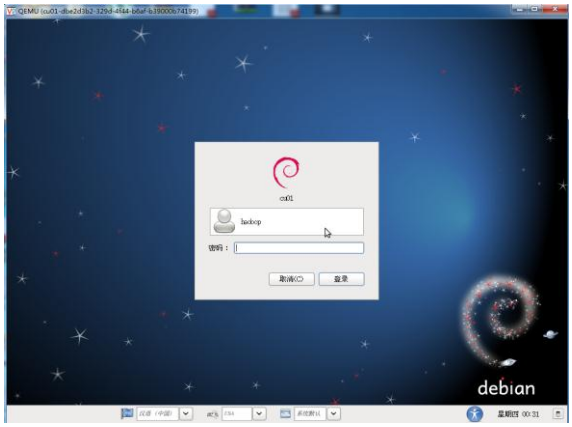

图 7 集群 A 中运行虚拟机作业

其次另选取 4 台上述配置节点，搭建集群 B。在其上搭建扩展了虚拟机 VM 这种应用的 YarnPlus（即修改后 hadoop-2.0.3-alpha）。搭建成功后安装 KVM。然后利用 YarnPlus 启动 hadoop-example 执行 WordCount，同时利用 YarnPlus 启动 kvm 虚拟机镜像，然后利用 VNC 连接该虚拟机，可正常使用，同时 WordCount 也顺利执行。

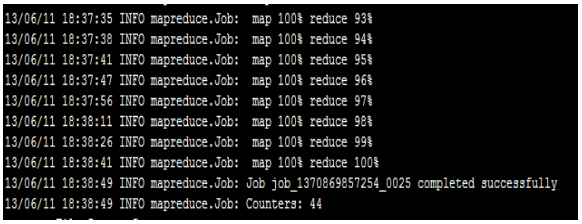

图 8 集群 B（YarnPlus）中执行 MapReduce 作业

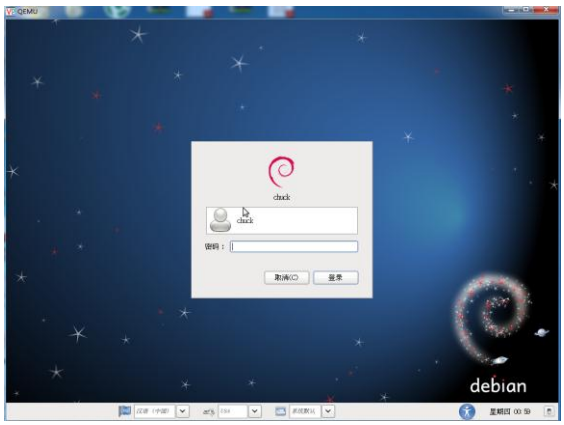

图 9 集群 B（YarnPlus）运行虚拟机作业

3.2 性能测试

主要设计实验进行简单混合共享方案（Hadoop merge VM），基于 VM 管理的资源共享方案（Hadoop on VM）和基于 Yarn 的资源共享方案（YarnPlus）的性能对比。首先对三种系统中同时运行 MapReduce 作业与 VM 时，两种异构任务的执行性能进行实验分析。最后设计实验分析 YarnPlus 系统在有 VM 任务运时进行大规模 MapReduce 数据处理的效率，并与 MapReduce 任务独占集群时数据处理效率进行对比。

在进行实验前选取性能测试的一些 benchmark，关于 Hadoop 任务主要如下表：

表 6 不同负载的资源敏感性

Workload	Resource sensitivity
Terasort	CPU+I/O
WordCount	CPU+Memory
PiEst	CPU

普通虚拟机服务对宿主机内存及分派客户机的内存资源最为关切，因此关于虚拟机性能通过一个矩阵链相乘负载的执行时间来进行测试。

首先进行系统中同时运行 VM 任务与 MapReduce 作业时两种异构任务的执行性能。利用集群 A 中 IVIC 系统创建 4 台高配置虚拟机（2cpu2g 内存）启动，然后在这四台虚拟节点中搭建与物理节点相同配置的 Hadoop1.0.4。同时在集群 A 中利用 IVIC 系统创建 25 台低配置虚拟机（1cpu512m 内存）启动。在低配置 VM 中运行矩阵链相乘，在虚拟 hadoop 系统中分别运行 Terasort、WordCount 和 PiEst 多次进行实验记录实验结构。然后关闭虚拟 Hadoop 集群，并在物理机 Hadoop 中运行 benchmark 测试，同时低配虚拟机继续执行矩阵链乘法记录实验结果。最后集群 B 启动 25 台低配虚拟机并同时运行 MapReduce job 记录实验结果。具体运行环境为：

表 7 任务运行环境

运行环境	解释
Hadoop merger VM	集群 A 启动 25 台低配虚拟机并在物理机运行 Hadoop 执行 MapReduce job
Hadoop on VM	集群 A 启动 25 台低配虚拟机和 4 台高配虚拟机，高配虚拟机运行 Hadoop 并执行 MapReduce job
YarnPlus	集群 B 利用 YarnPlus 启动 25 台低配虚拟机，同时利用 YarnPlus 运行 MapReduce job

MapReduce 任务实验结果如下：

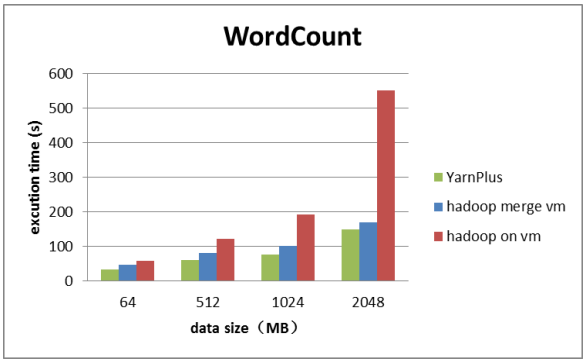

图 10 不同数据大小 WordCount 运行时间对比

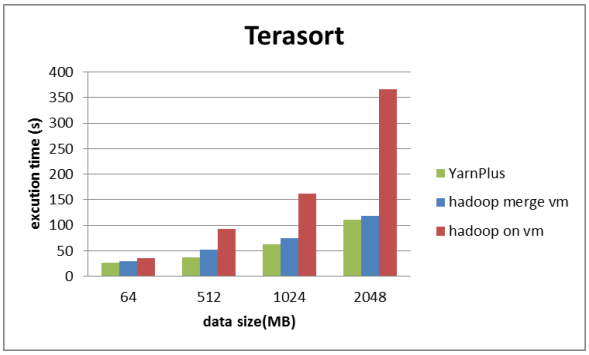

图 11 不同数据大小 Terasort 运行时间对比

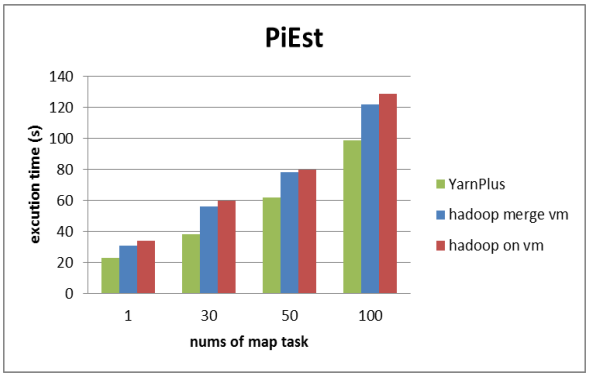

图 12 不同 task 数目 PiEst 运行时间对比

通过以上实验可以看出：第一，YarnPlus 中三种 MapReduce 任务执行所消耗时间最少，性能最好。这主要是由于 YarnPlus 中由 RM 根据整体资源使用情况在 VM 与 MapReduce Job 间合理分派资源，避免了 MapReduce Task 与 VM 抢占资源影响任务性能。第二，针对以 Terasort 和 WordCount 为代表的数据处理，虚拟 Hadoop 处理所需时间明显高于物理 Hadoop，而且随着处理数据规模的增大，虚拟 Hadoop 所消耗时间变得难以接受。这主要是由于虚拟机中 I/O 虚拟化导致，为了满足多个客户机操作系统对外设访问的需求，虚拟机监控器 VMM 以模拟的方式向客户机提供外设的访问，即截获客户机操作系统对设备的访问请求，然后通过软件模拟真实物理设备。这种模式使得虚拟机 HDFS 中的数据访问变得异常繁琐，当不满足数据本地性时，数据块的获得要通过多次设备模拟来取得（网络设备，磁盘设备），这造成了任务执行时严重的时间开销，而对于 PiEst 这种只消耗 CPU 的任务性能间的差距不是很大。

其中 VM 中负载运行结果如下：

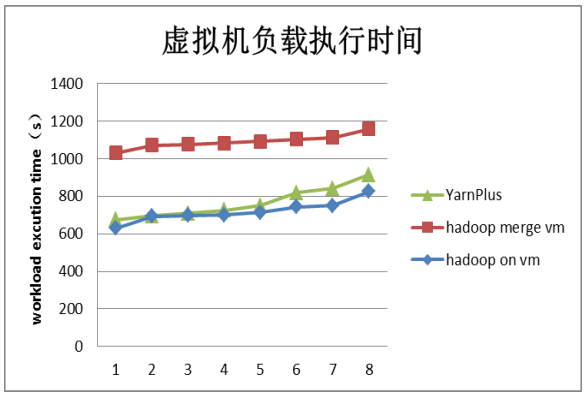

图 13 VM 负载运行时间对比

通过对 VM 运行矩阵链相乘负载执行时间在三种系统中进行对比。可以看出在有 MapReduce job 运行的时候会对 VM 性能产生一定影响，但是在 YarnPlus 中由于对 VM 和 MapReduce 任务统一管理，合理分派资源避免了产生资源冲突现象，所以性能开销很小，只有 7%（由于 hadoop on vm 环境中只有虚拟机，因此可将此时系统中负载执行时间视为 IaaS 环境中该负载执行时间），而在没有进行统一管理的共享系统中，虚拟机性能开销超过 50%。

最后，由于处理数据集较小时（数据规模小于 2GB），Hadoop merge VM 与 YarnPlus 性能差距不大，现进行实验比较两种系统处理较大规模数据集（数据规模 7GB 与 15GB）时 MapReduce 执行性能，同时对结果与 Hadoop 独占集群资源时任务执行性能进行对比。

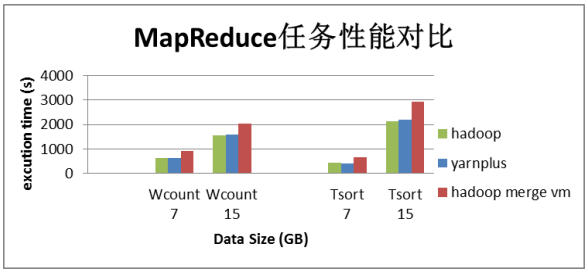

图 14 大数据集数据处理运行性能对比

以上结果表明在同时启动 MapReduce job 和 VM 任务的情况下进行大规模数据处理时，YarnPlus 系统与 Hadoop merge VM 系统相比，在处理 7GB 数据时，WordCount 任务执行时间缩短 29%，TeraSort 任务执行时间缩短 27%，当处理 15GB 数据时两类任务执行时间分别缩短 21%与 27%，即相比简单混合系统 YarnPlus 中数据处理能力明显提高。同时可以发现 YarnPlus 系统 MapReduce 任务数据处理效率与 MapReduce 任务独占集群资源时基本一致。这主要因为：第一，利用 YarnPlus 将虚拟机管理扩充为 Yarn 中一类任务后，Yarn 在资源分派时可以同时考虑虚拟机与 MapReduce Job 的资源需求，从而达到各个节点负载均衡，既避免某些节点负载过高而拖累整体任务执行进度，也避免了由不同任务抢占同一资源产生的资源竞争。第二，是由于 Yarn 对 task 所使用资源封装在 Container 中，并采用了 Linux Container 和 OS 线程监控的隔离技术，使得 VM 对 MapReduce Job 的性能干扰减少。

4 结论及未来工作

综上所述，利用 Yarn 对其扩展来支持 VM 与 MapReduce job 间的资源共享，可以让 Yarn 的调度器同时获得集群中两类任务的资源需求和集群真实负载状况，这样避免了不同任务队同一资源竞争而产生性能下降，不仅能保证 VM 服务的性能，而且相比简单混合系统缩短了 MapReduce Job 的执行时间，高效地实现了异构任务间的细粒度资源共享，从而避免了多个集群的部署，将集群所耗机器数目减少，从而提升了资源利用率，节省了成本。

目前，扩展后的 Yarn 系统 YarnPlus 是可用的，但对多任务之间的资源分派仍然采用 Hadoop 中默认的调度算法，需要更加详细的分析这两类任务运行的 Trace log 来设计更加适用于这两种任务混合调度的算法。另外，可向用户提供一个友好的可视化界面，使用户在网页即可方便的提交任务并监控任务执行进度。以上两点是后续的主要研究内容。

参考文献

[1] J.Dean and S.Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters.” OSDI 2008

[2] Hadoop http://hadoop.apache.org/

[3] M. Zaharia et al. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. EuroSys 2010.

[4] J. Polo, D. Carrera, Y. Becerra, V. Beltran, and J. T. andEduard Ayguad. Performance management of accelerated mapreduce workloads in heterogeneous clusters. ICPP 2010.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph,R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson,A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department,University of California, Berkeley, Feb 2009.

[6] Mark Yong, Nitin Garegrat ,Shiwali Mohan.“Towards a Resource Aware Scheduler in Hadoop”ICWS 2009

[7] Y.Yu,P.K.Gunda,et al. Distributed aggregation for data-parallel computing：interface and implementations . SOSP 2009.

[8] Tom White, Hadoop: The Definitive Guide, 2010

[9] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, and Randy Katz Ion Stoica, “Improving MapReduce Performance in Heterogeneous Environments”OSDI 2008

[10] M. Zaharia，D. Borthakur，J. Sen Sarma，S. Elmeleegy，S. Shenker，and I. Storica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling .In EuroSys 2010

[11] Mesos http://incubator.apache.org/mesos/

[12] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R. Katz, S. Shenker, I. Stoica. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. NSDI 2011.

[13] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski,S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. NSDI 2011

[14] K. Thirupathi Rao, P. Sai Kiran, Dr. L.S.S Reddy, V. Krishna Reddy, B. Thirumala Rao, “Genetic Algorithm For Energy Efficient Placement Of Virtual Machines In Cloud Environment”, in proc IEEE International Conference on Future Information Technology (IEEE ICFIT 2010), China, December 2010, pp: V2-213 to V2-217

[15] Apache. Hadoop profiler: Collecting cpu and memory usage for mapreduce tasks. https://issues.apache.org./jira/browse/MAPREDUECE-220

[16] A.Ghodsi and et al. Dominant resource fairness:fair allocation of multiple resource types. NSDI 2011

[17] B.Thirmala Rao, LSS.Reddy. Survey on Improved Scheduling in Hadoop MapReduce in Cloud Environments. International Journal of Computer Applications Vol. 34, No. 9, Nov 2011, pp.29-33

[18] Andrew Konwinski. Multi-agent Cluster Scheduling for Scalability and Flexibility. Technical Report No. UCB/EECS-2012-273.EECS Department, University of California, Berkeley. 2012

[19] Apache Hadoop NextGen MapReduce (Yarn). http://hadoop.apache.org/docs/current/hadoop-Yarn/hadoop-Yarn-site/Yarn.html

[20] Map-Reduce 2.0. https://issues.apache.org/jira/browse/MAPREDUCE-279

[21] Jinpeng Huai, Qin Li, Chunming Hu. CIVIC: A Hypervisor Based Virtual Computing Environment. HPCC 2007

[22] Kivity, Y. Kamay and D. Laor “kvm: the Linux Virtual Machine Monitor”, in Proc. Linux Symposium vol. 1, pp. 225-230, 2007.