A Dissertation Submitted to Zhejiang University for the Degree of Master of Engineering

TITLE: A Design and Implementation of High-performance and Large-scale Grid Monitoring System base on Cloud Architecture

Author: Xiangyu Zhang

Supervisor: Vice-Professor Huajun Chen

Vice-Professor Xiaohong Jiang

Subject: Computer Architecture

College: Computer Science and Technology

Submitted Date: January, 2012

摘要

网格系统通过一体化架构、规范化接口、标准化服务等手段，实现了计算资源、存储资源、服务资源、数据资源等各种资源的集成共享和跨域协同，充分利用互联网上大量的闲置资源。网格环境下各类资源具有大规模，异构以及地域分布广泛等特点，资源可以动态的加入或者离开网格虚拟组织，构建一个高性能大规模的全局监控系统具有非常重要的意义。

为了满足网格监控系统对高可用性，高可扩展性，高实时性和高效性等需求，我们提出了一个基于云架构的高性能大规模网格监控系统。本文介绍了该系统的详细设计与实现，并着重介绍了海量监控数据存储和资源聚合两个关键技术。

本文采用了基于MongoDB的海量存储技术来存储资源状态监控数据，解决了以往监控系统只能保持最近一段时间的监控数据，并且监控数据孤立，不能产生更多价值的问题。该存储架构提供丰富灵活的数据模型，能够支持高并发的读写请求，能够提供高效的存储和访问，能够提供丰富的查询和检索接口，有良好的可扩展性，能适应集群规模的不断增长。本文还通过采用基于一致性哈希的高效易扩展资源聚合技术，解决了以往资源聚合技术采用树形结构带来的实时性低，易出现单点故障等问题。该聚合技术能够自动的分配资源与索引服务器的对应关系，易于实现负载均衡，增加或者减少索引服务器对已有结构改变小，同时能够自动检测和处理故障机器。

关键词： 网格资源监控，海量数据存储，资源聚合，MongoDB，一致性哈希

Abstract

Grid computing system makes the resource sharing among virtual organizations and cross domains possible. It integrates multiple distributed resources, containing computing resource, storage resource, services and data. As the resources in grid environment are distributed, heterogeneous, large scaled and the resources can attend or leave the virtual organizations dynamically, it is so important to construct a global monitor system, which can monitor the running state of the system, help load balancing and predict the bottleneck of the running system from analyzing the monitoring data.

It's necessary for the gird monitoring system to be high available, high scalability, real time and efficient. We use the cloud computing architecture to meet these demands. This paper give the design and implementation of the high-performance and large-scale grid monitoring system base on cloud architecture.

Existing monitor systems have the problem that, only the monitor data of last few hours can be maintained and the monitor data stored isolated. Our monitoring system uses one of the NoSQL database MongoDB to store the monitor data, which have many benefits, containing agile, scalable, high availability and support abundant ways of query and search. We use a resource aggregate framework based on consistent hashing, which is efficient, high scalability and load balance.

Keywords: Grid resource monitor, big data storage, resource aggregate, MongoDB, consistent hashing

目录

图目录

表目录

第1章 绪论

1.1 课题背景及研究意义

随着计算机与网络技术的飞速发展，互联网技术迎来了海量数据的时代，越来越多的大规模集群被组建起来，云计算、网格计算等技术成为了人们研究的热点。网格技术旨在基于互联网实现包括计算资源、存储资源、服务资源等的跨单位协同共享、跨地域任务协同、一体化资源管理和按需应用服务等能力[1]。网格计算技术经过近二十年的发展，其整体技术体系已经非常成熟，并在高能物理、对地观测、生命科学、天文学、气象学等领域得到广泛的应用。网格是系统之系统，是构架于各系统之上的大系统，它的构建基本原则是尽可能的不破坏已有系统的结构和运行，通过一体化架构，规范化接口，标准化服务等手段解决跨系统的集成共享，跨域协同和全局监管等功能。网格通过互联网把地理位置分散的计算存储资源联合起来，协同工作，来完成单台机器无法完成的大型复杂任务[2]。

网格监控系统能够为网格系统提供完善、良好的运行和操作环境的基础，承担对网格系统的软硬件设置及服务的运行状态进行监视和管理任务；提供异构的软硬件资源的实时监控和分时、分组的动态任务分配情况，并且通过对各资源共享平台的基础设施的元数据的获取、聚合和发布，提供基于元数据的更新和查询。对于分布异构的网格环境来说，需要监控网格内跨组织或跨管理域的各种大量异构资源的状态，以及对这些资源的软硬件使用情况，为用户提供软硬件资源的实时状态信息，使得管理员用户能够及时掌握网格系统的运行状态，迅速发现系统运行的瓶颈所在，为作业调度与分配系统提供机器的运作情况，使得作业调度与分配系统能够根据机器的负载状况信息，找到合适的机器来执行作业，使得系统资源充分利用。

由于网格监控系统要对网格系统中大规模的集群系统，海量数据以及大量网格服务和任务调度状态信息进行监控，这些监控信息数据量大，更新频率快，因此对监控信息的存储系统有高可用性，高可扩展性的需求，需要能够处理海量数

据，并且提供丰富高效的查询检索接口。而监控数据对实时性要求能高，需要能够尽快及时的传递到用户面前，才能使用户了解到系统当前的运行状态，才能使作业调度分配模块能够及时的掌握系统负载状态，因此监控系统对高速的海量监控数据聚合技术有很高的要求。根据这些需求和特点，我们提出基于云架构来解决监控状态数据存储以及监控数据聚合的问题，构建了一个高性能大规模的网格监控系统。

1.2 需求分析

云计算技术通过大量廉价的硬件资源，解决了海量数据存储，可扩展性，可靠性和可用性等问题。网格资源监控系统对云架构技术的需求可分为以下几点：

- 网格监控系统对海量数据存储技术的需求

网格系统需要调度和调配大量分布式异构资源，以满足网格内各个虚拟组织和用户的各种大规模计算存储任务需求。网格监控系统需要提供网格系统中地理上分散的各类软硬件资源的实时运行状态，提供作业执行和调度信息，使得网格系统的资源能够充分的利用，用户的任务可以尽快的完成；还需要能够通过网格系统运行的历史数据，评估网格系统一段时间内的运行状态，预测各个节点已经系统今后的运行的负责情况，为网格系统的优化和建设提供依据，因此需要对网格监控系统获取到的网格监控状态信息进行持久化存储，而由于网格系统包含大量资源，资源不断的动态变化，资源状态信息刷新频率快，数据量大，因此需要海量数据存储技术来存储海量监控状态数据。

- 网格监控系统对高可扩展性的需求

网格系统为网格内各个虚拟组织和用户提供网格的各种资源，需要网格具备高可扩展性来适应用户量的增长以及虚拟组织和用户对于资源需求的增长。而网格监控系统要实时获取收集和存储网格系统的监控状态数据，因此同样需要具有高可扩展性，这样才能适应网格系统的变化。而网格监控系统的高可扩展性体现在多个方面，首先网格监控系统需要从海量资源中获取和收集资源状态数据，对收集到的海量监控数据进行聚合，需要聚合框架能够良好的伸缩，适应网格系统

规模的扩大。其次，网格监控系统需要对海量监控数据进行存储，而且监控数据的存储随着时间的增长而积累，所以对网格监控数据存储系统的高可扩展性也有很高的需求。

- 网格监控系统对灵活的数据存储模式的需求

网格系统中所包含的各类资源类型丰富，并且经常有新的资源加入网格系统，这就需要网格监控系统能够灵活的扩展数据类型和数据结构模型，以适应新加入的各种资源。而传统的关系型数据库很难满足这个需求，因此需要基于云架构的无模式的海量数据存储技术来解决灵活数据模式的问题。

- 网格监控系统对高可用性和可靠性的需求

网格系统包含海量的资源和机器，而在这样大规模的资源面前，某些机器出现故障是不可避免的，因此需要网格监控系统具有高可用性和可靠性，不能因为单点故障的出现，而导致大量的资源和机器无法进行监控。网格监控数据存储系统也需要能够通过备份冗余等多种方式，来保证系统的高可用性和可靠性，保障监控数据的安全。

1.3 本文贡献及创新点

本文的主要贡献及创新点有：

- 本文通过对网格监控系统的实现难点进行分析，提出了基于云计算技术来解决网格监控系统对于海量数据存储，高可扩展性，高可用性和可靠性的需求，实现了一个基于云架构的高性能大规模网格监控系统。
- 本文提出了使用基于NoSQL技术的MongoDB数据库，来为网格监控系统存储海量的监控数据。解决了以往监控系统只能保持短时间的监控数据，并且监控数据孤立，不能产生更多价值的问题。而该存储架构能够支持高并发的读写请求，能够提供高效的存储和访问，能够提供丰富的查询和检索方案，很好的满足了网格监控数据存储的需求。
- 本文提出了基于一致性哈希的高效易扩展资源聚合技术，解决了以往监控系统中的资源聚合框架采用树形架构带来实时性低，容易出现单点故

障，不容易进行扩展的问题。该聚合技术能够自动的分配资源与索引服务器的对应关系，易于实现负载均衡，增加或者减少索引服务器对已有结构改变小，同时能够自动检测和处理故障机器。

- 本文通过使用一些富客户端技术，以图表，地图等多种丰富的展现形式，对网格系统，集群，节点的运行状态，任务的完成状态等进行可视化展示。使得用户可以更加清晰，直观的从中获取有用的信息。解决了以往大多数监控系统使用 RRDTool 绘制图形，对监控数据的展示方式不够灵活等缺点。

1.4 论文结构

第一章首先介绍了课题的背景以及研究意义，分析了网格监控系统对基于云架构技术的需求，总结了本文的创新点，最后对整篇论文的结构进行了介绍。

第二章介绍了网格监控系统的国内外研究现状，对比较有影响力和代表性的网格监控系统进行了介绍。接着对本系统用到的相关技术进行了介绍。

第三章介绍了基于云架构的网格监控系统的整体架构设计。首先分析了网格监控系统应该满足的总体要求，以及应该实现的功能。接着给出了网格监控与管理系统的体系架构图，并分别介绍了资源层，聚合层，服务层，展示层各层的主要功能和作用。

第四章首先对资源监控数据的特点进行了分析，列举了采用 MongoDB 数据库存储资源监控数据的优势。接着给出了基于 MongoDB 的海量监控数据存储模块的总体设计，又从多个方面介绍了该模块的详细设计与优化方法。最后对 MongoDB 进行了一些性能测试和分析。

第五章首先简要介绍了一致性哈希算法以及对一致性哈希算法的 chord 实现。然后分析了现有资源聚合框架 MDS 存在的问题。根据 MDS 的不足，我们给出了基于一致性哈希的高效易扩展资源聚合模块的总体设计，并详细介绍了该模块的具体设计与实现。

第六章介绍了基于云架构的网格监控系统的详细设计与实现。首先介绍了资

源层的监控对象与事件，以及资源采集模块的详细设计与实现。接着介绍了聚合层中资源注册发布模块的设计与实现，以及聚合模块是资源的订阅的实现。最后对展示层的实现进行了详细介绍，介绍了展示层使用的主要技术，详细的描述了展示层实现的难点，并给出了系统的可视化效果图。

第七章是工作总结与展望，对全文的工作进行了总结，并提出了系统未来需要继续完善的工作。

1.5 本章小结

本章首先介绍了课题的背景、需要解决的主要问题、研究的意义，分析了网格系统对网格监控系统的需求以及网格监控系统应该包括的功能，论述了本文的主要工作已经创新点。最后介绍了论文的结构以及每个章节的主要内容。

第2章 研究现状及相关技术

2.1 网格监控系统的国内外研究现状

2.1.1 网格监控体系结构(GMA)

网格监控体系结构（GMA, Grid Monitoring Architecture)[3]，是全球网格论坛（Global Grid Forum）认定的网格监控系统的标准体系结构。网格监控系统需要有高可扩展性，高可用性，能够处理分布式的海量资源数据，能够聚合来自分散地理位置的异质异构的资源数据，提供网格资源的整体状况和实时信息。

网格监控体系结构包含三类组件：

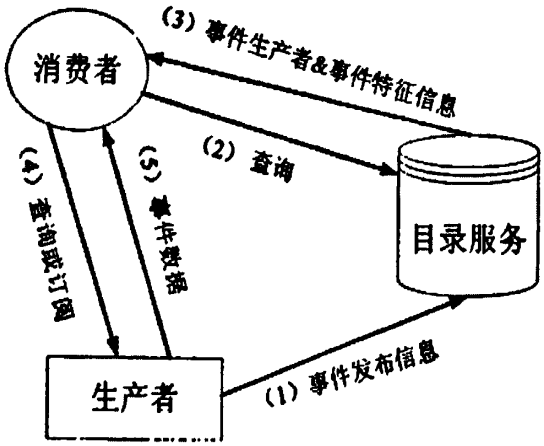

图 2.1 网格监控体系结构图

- 目录服务：支持信息的发布和发现；当生产者和消费者在目录服务中注册时，通常也描述了他们产生和消费的数据类型。他们可以给一些数据设定静态数值，可以限定数据的范围。这些发布的信息可以使其他的生产者或者消费者发现可用的数据类型，数据的源和汇等。目录服务不负责数据的存储，只负责数据的模式信息等。

一个目录服务提供的服务有：

1) 授权修改——客户端可以增加，修改，删除注册项

2) 授权查询——客户端可以对注册项进行查询

- 生产者：产生性能数据（源，source）。生产者通过事件发布的方式，将其可以提供的数据类型以及获取数据的方法，周期等信息注册在目录服务中，消费者可以再目录服务中查找和发现已经注册过的监控信息源（生产者），进而与监控信息源通信，获取相应的监控数据
- 消费者：接受性能数据（汇，sink）

消费者需要能够支持一下功能：能够订阅和取消生产者的数据，能够对查询目录服务并进行相关的操作。

2.1.2 资源监控与发现服务(MDS)

Globus[4] Toolkit 提供了一个完整的开放网格服务基础设施实现。监控与发现服务 MDS（Monitor and Discovery Service）[5,6]是 Globus Toolkit 的信息服务（Information Service）的主要模块。实现了一个元数据聚合框架，提供元数据的注册，发布，聚合，查询，访问等多种机制。MDS 的最新版本是 MDS4，它符合 WSRF(Web Service Resource Framework)规范，可以监控各类资源以及 Web Service 的状态。MDS 的主要组成部分为：

- 索引服务（Index Service）

索引服务主要负责从网格资源中收集监控信息，并进行统一的发布。通常来讲，一个虚拟组织会部署一个或者多个索引服务，这些索引服务会收集这个虚拟组织能够访问的所有资源的状态信息。收集到的信息通过 WS-Resource 属性来进行发布。

- 触发器服务（Trigger Service）

管理员预先定义一些规则和阈值，如果触发器服务发现系统运行状况满足规则，则触发管理员预先定于的操作。例如当等待处理的作业队列长度带到一定值时，给管理员的手机发送一条短信。触发器服务通过订阅一些 WS-Resource 属性来实现这个触发功能。

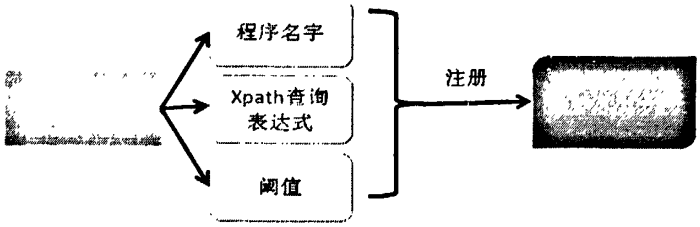

图 2.2 注册触发服务示意图

- 聚合框架：

聚合框架是 Globus MDS4 中用于收集和聚合数据和资源的软件框架。它提供统一的聚合源接口，可以收集 XML 格式的数据，并有统一配置机制来配置聚合源的选择和相关的参数（获取什么数据，从哪里获取数据）。元数据聚合框架通过 WSDL 定义了一个聚合服务小组类型，包括配置信息和数据。管理客户端使用标准的 WSRF 服务组注册机制，来注册这些服务组项到聚合服务。聚合框架采用观察者模式，资源为被观察者，订阅资源的用户为观察者，通过实现相应的接口，当资源状态属性发生变化时，自动通知给订阅资源的用户。

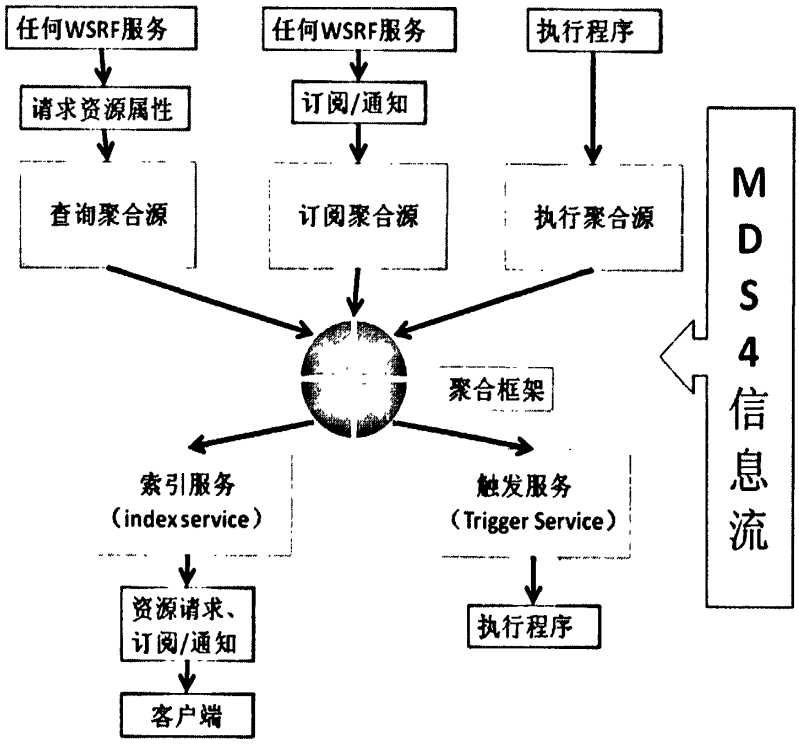

图 2.3 MDS4 的系统结构图

2.1.3 基于关系的网格监控体系结构(R-GMA)

R-GMA[7,8]是一个网格信息服务和监控系统，由欧洲数据网格项目支持。R-GMA 在 GGF 提出的 GMA 网格监控体系结构的基础上进行了一些改进，它采用了传统的强大灵活的关系模型来实现。它拥有两个特殊的属性：

- 任何人向 R-GMA 提供或索取信息时不需要了解注册器，消费者和生产者在幕后处理注册事务；
- 资源监控系统类似于一个巨大的关系型数据库，并可以以类似方式进行查询处理。

R-GMA 提供了一种在网格环境中使用关系模型的方法，并且不是一个一般的分布式 RDBMS。所有的信息生产者之间独立。在关系型这个意义上，生产者通过一个 SQL CREATE TABLE 声明通告其必须发布的信息。生产者使用 SQL INSERT 发布信息，消费者使用 SQL SELECT 获得他们需要的信息

2.1.4 分布式监控框架(DMF)

分布式监控框架 DMF[9]（Distributed Monitoring Framework）是由美国 Lawrence Berkeley 国家实验室的数据密集分布式计算研究小组开发的。这个分布式监控框架主要是针对高速广域网环境以及数据密集型应用的。目前采用 DMF 框架的项目有：EU DataGrid，Particle Physics Data Grid 等

DMF 的主要组成部分有：

- Instrumentation：DMF 提供工具可以方便的向网格中间件添加指令，并以一种标准的方式发布这些事件数据。
- Sensors：包括网格的传感器，主机传感器等。DMF 定义了一套标准的 schem 和发布机制来表示和发布这些传感器收集的数据。
- Event publication：为了处理海量的传感器收集的数据，DMF 提供一种灵活的，高可扩展性的方式来发布和订阅服务。
- Event archives：事件数据的归档对于性能分析和性能调优有非常重要的作用。DMF 提供高性能可扩展的数据存储方式和数据分析工具 NetLogger。

2.1.5 其他网格监控系统

- GridICE[10]，是一个为网格系统设计的分布式监控工具。它促进了标准网格信息服务接口、协议以及数据模型的采用。提供多中粒度的监控，既可以查看网格整体运行状况，又可以查看资源的状态细节，并且能够提供资源的完整的历史监控数据。
- MapCenter[11]，是一个开源的网格状态可视化工具，提供了灵活的数据展示方式。
- GridEye[12]，一个面向服务的网格监控系统，对网格资源进行统一的描述和表示，能够支持多种硬件资源，服务资源的监控，以及历史数据统计和负责预测。
- GridRM[13]是一个开源的通用监控系统，为各种客户端提供一个将收集到的数据转换成符合各客户端要求的数据工具，可以用来监控广泛分布资源。GridRM 对添加新的监控的配置复杂，系统扩展性不够好。

2.2 相关技术介绍

2.2.1 Ganglia 集群监控技术

Ganglia 是由加州大学伯克利分校开发的一个可扩展的分布式监控系统，用于网格环境下的计算资源、存储资源、网格资源和服务资源的监控。其结构为树状层次结构，因此它有着很好的可扩展性，可以很容易地适应较大规模的集群[14]。

Ganglia 采用 XML 用于数据描述，XDR 做压缩，轻量级的数据传输，RRDtool 做数据存储和可视化展示如下图所示，每台计算机都运行一个收集和发送度量数据（如处理器速度、内存使用量等）的名为 gmond 的守护进程。它将从操作系统和指定主机中收集资源的状态信息。接收所有度量数据的主机会运行一个名为 gmetad 的守护进程，可以显示这些数据并且可以将这些数据的精简表单传递到层次结构中。正因为有这种层次结构模式，才使得 Ganglia 可以实现良好的扩展。最后用户通过 Ganglia PHP Web Frontend（基于 web 的动态访问方式）即可实时

获取监控信息。Ganglia 带来的系统负载非常少，这使得它成为在集群中各台计算机上运行的一段代码，而不会影响用户性能。使用 Ganglia 超大规模集群监控软件监测到的性能评测数据，是进行良好网格管理的基础，也有利于系统或用户做出最优的调度决策。

Ganglia 主要由数据采集器、数据收集器、web 可视化工具和额外扩展工具 gmetric 五部分组成。数据采集器名叫 gmond，是 Ganglia Monitor Daemon 的缩写。gmond 作为一个守护进程部署于各台监控机器上，用以收集各台监控机器的信息。数据收集器名叫 gmetad，是 Ganglia Metadata Daemon 的缩写。它通过轮询收集 gmond 的数据，并聚合各类信息，然后保存在本地的 rrdtool 的数据库中。多个 gmetad 可以构成多层监控架构，所以通常每个集群都有一个 gmetad 节点，如图 2.4 所示。

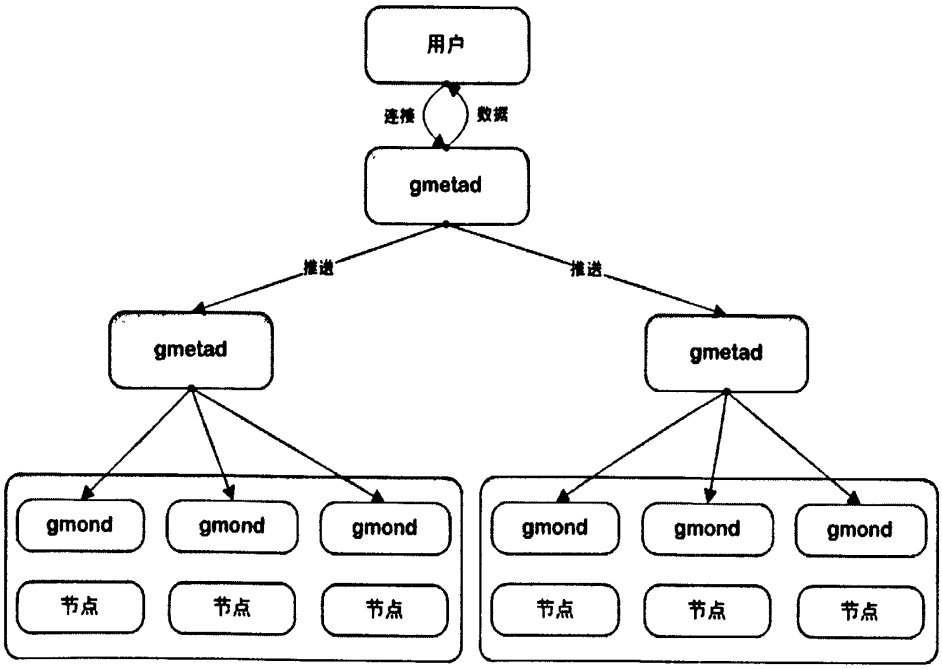

图 2.4 Ganglia 系统架构图

Ganglia 的 Web 可视化是用 PHP 脚本加 rrdtool 实现的。它以曲线图为主，以时间为主线反映各种监控信息，可以部署到任何支持 PHP、SSL 和 XML 的 web 服务器，一般都使用 Apache2。如图 2.4 是 UC Berkeley 大学部署的 Ganglia 集群

监控系统的效果展示。

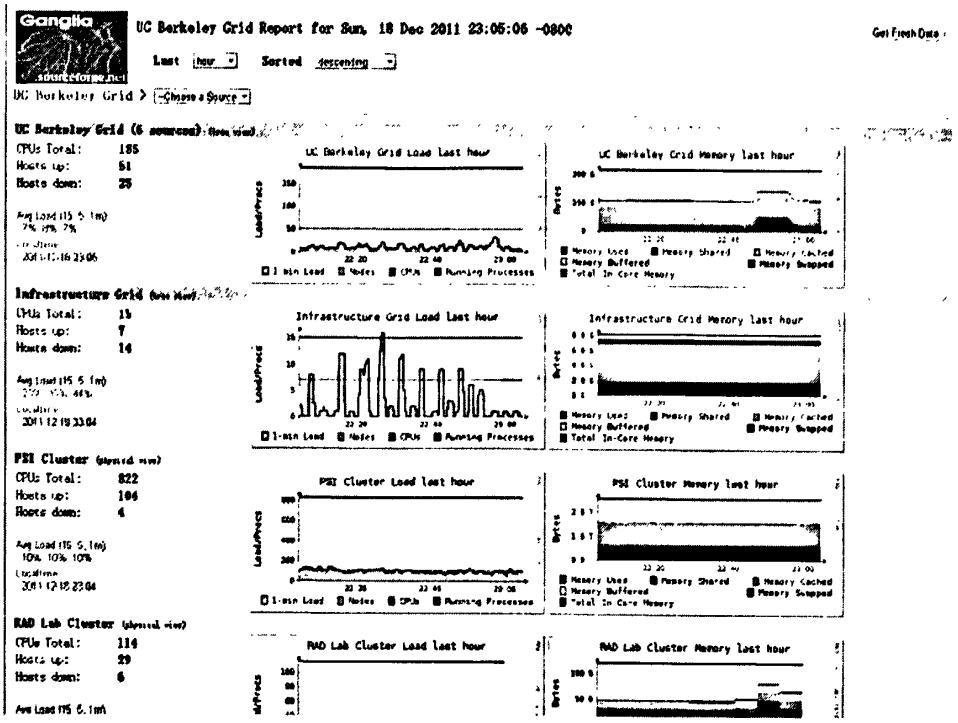

图 2.5 Ganglia 监控系统可视化效果图

rrdtool(Round Robin Database Tool)[15]是一个强大的绘图的引擎，很多工具例如 MRTG 都可以调用 rrdtool 绘图。rrd 是指环状数据库是一个主要用于进行监控的数据库，与其它的数据库相比，具有以下 4 个特点:

1) 除了存储数据之外，它具有可以创建图形的工具；
2) 它的数据库文件大小是固定的，新的数据添加到已有数据的后面，当到了文件末尾的时候就开始从文件开始写数据，Round Robin 就是指这个意思；
3) 一般的数据库只能存储数据本身，而 rrd 可以存储相对与以前的数据的变动；
4) 一般的数据库是在提供数据的时候才更新，而 rrd 是在每一个预先设好的时间间隔都会更新，每次更新的时候，time stamp 也会存储进去。

如果想使用 Ganglia 监控并绘制没有包括在标准的 Ganglia 发行版中的性能量度，可以使用 gmetric 编写自己的性能监视脚本报告运行在集群节点本地的

gmond。对于不能直接运行 gmond 的设备，也可采用 gmetric 来发送数据。Ganglia 可篡改来自其他主机的报告，因此 Ganglia 可监控来自嵌入式系统、硬盘和其他设备的数据。例如，很多 UPS 有内置温度传感器，并可通过建立 SNMP 代理来支持网络管理卡。假设你有 Net-SNMP 项目的 snmpget，就可以编写脚本将 UPS 的温度数据传给 Ganglia。

2.2.2 基于 NoSQL 的海量数据存储技术

随着互联网的发展，互联网应用的用户量剧增，需要处理成百上千万用户提交的海量数据，对海量数据存储技术提出了新的要求。然而传统的关系型数据库难以提供超大规模的数据存储以及高并发的读写能力。并且关系数据库的一些主要性能，如事务一致性、复杂的 SQL 查询特别是多表关联查询能功能，在现代互联网应用中显得并不是十分重要。

自从 Google 提出 BigTable 这个非关系型数据库模型后，NoSQL 技术逐渐成为业界探讨和人们研究的热点。出现了 HBase，HyperTable，MongoDB 等一系列基于 NoSQL 的存储技术，弥补了关系数据库的不足。

网格监控系统会收集到海量的物理资源状态信息，数据资源元数据信息，作业队列信息以及作业调度信息等数据资源，对海量的网格监控信息进行高效的存储，查询和访问，一直是网格监控系统中的一个难题。我们采用基于 NoSQL 的海量数据存储技术，来达到相应的要求。NoSQL 非关系型数据库有如下特点：

1) 支持高并发的读写请求。

2) 支持海量数据高效的存储和访问。

3) 具有高可扩展性和高可用性。

MongoDB[16]是一个面向文档的数据库，由 10gen 公司开发并维护，是非关系数据库中功能最丰富的，主要解决海量数据的访问效率问题。MongoDB 使用类似 JSON 风格的语法，支持复杂的数据结构；不需要预先定义模式(schema-less)，支持嵌入文档，非常灵活方便。并有 MongoDB 有强大的数据查询功能，几乎可以实现类似关系数据库单表查询的绝大部分功能。根据 MongoDB 的官方文档中

介绍，当数据量达到50GB以上时，MongoDB的访问速度是MySQL的10倍以上。MongoDB对事务的支持较差，不支持事务级别的锁定，由于监控数据用来做统计和分析，对数据安全性的要求并不是非常高，所以MongoDB依然是非常适合的选择。

2.3 本章小结

本章首先介绍了网格监控系统的研究现状，接着介绍了本系统用到的Ganglia集群监控技术以及NoSQL海量数据存储技术。

第3章 基于云架构的网格监控系统的总体设计

3.1 总体要求与功能

3.1.1 总体要求

- 实时性：由于监控系统需要实时监控整个网格的状态，监控数据的生命周期短，更新频率快，对数据的时效性要求高，所以要求监控系统有较低的延时，以保证能够实时有效的掌握网格系统各种资源的最新状态。当系统负载不平衡或者出现故障时，要能迅速的发现和定位瓶颈所在。
- 高效性：网格监控系统对整个网格状态的获取和收集不能对系统的性能，网络以及服务产生明显的影响，要求网格监控系统自身高效，稳定，运行负载低。
- 可扩展性：由于网格是动态的，节点可以随时加入和删除，需要系统有很强的可扩展性，可以适应网格规模和数量上的快速增长。同时，网格资源是异构的，要求监控系统有良好的体系结构，要能够在不改变已有系统的前提下方便的添加新的监控项。
- 可靠性：网格监控系统要为网格系统提供可靠的监控服务，保证资源监控数据的获取，聚合、存储和展示等各个方面的可靠性，能够自动发现和解决系统中机器出现的故障，保证系统稳定的运行。

3.1.2 功能设计

网格监控系统应该包含以下功能：

- 提供全局资源视图。网格系统具有分布、异构、动态等特点，网格资源监控与管理系统要为网格的资源、服务、数据提供一个统一的注册和发布接口，提供一个全局资源视图，用以屏蔽地域分布广泛，节点可以动态加入和删除的海量计算存储资源，使之呈现为一个统一的整体。

- 获取网格资源的静态属性信息以及动态监控信息。提供网格资源的注册与发布接口，实时动态获取网格资源的状态信息，为网格监控，性能分析，已经作业的动态调度提供数据基础。
- 收集存储网格资源的状态信息。由于网格系统规模非常巨大，服务于网格系统的机器数量庞大，收集到的监控数据信息是大规模海量的，因此需要一个高效的，高可靠性，强可扩展性的存储方案，来为监控数据提供持久化存储，为其他需要监控数据的模块提供功能丰富的查询和访问接口。
- 提供对资源状态的订阅功能。由于大部分用户只对部分资源的信息感兴趣，要提供可定制的订阅功能，可以订阅部分感兴趣的数据。当数据发生变化时，能够及时，高效的通知给订阅者。
- 提供资源监控的可视化展示。由于监控数据信息量非常大，用户从海量数据中很难获得有用的信息，而通过对监控数据进行可视化，用图表的方式将监控数据展示出来，可以很清晰的看出监控系统中集群和机器的运行状态以及发展趋势，可以明显的了解到机器的负载情况。
- 提供可靠的告警机制。监控与管理系统的最重要的功能之一就是监控系统状态，当系统运行状态或者负载情况异常时，及时采取相应的措施，并通知运维管理人员，并为运维管理人员提供尽可能详细、准确的日志信息，可用于快速定位系统运行的瓶颈以及故障所在，保证系统平稳稳定的运行。

3.2 系统总体架构

本系统实现了一个基于云架构的高性能大规模的网格监控系统，系统分为资源层、聚合层、服务层和展示层四个部分。具体体系结构图如图3.1所示：

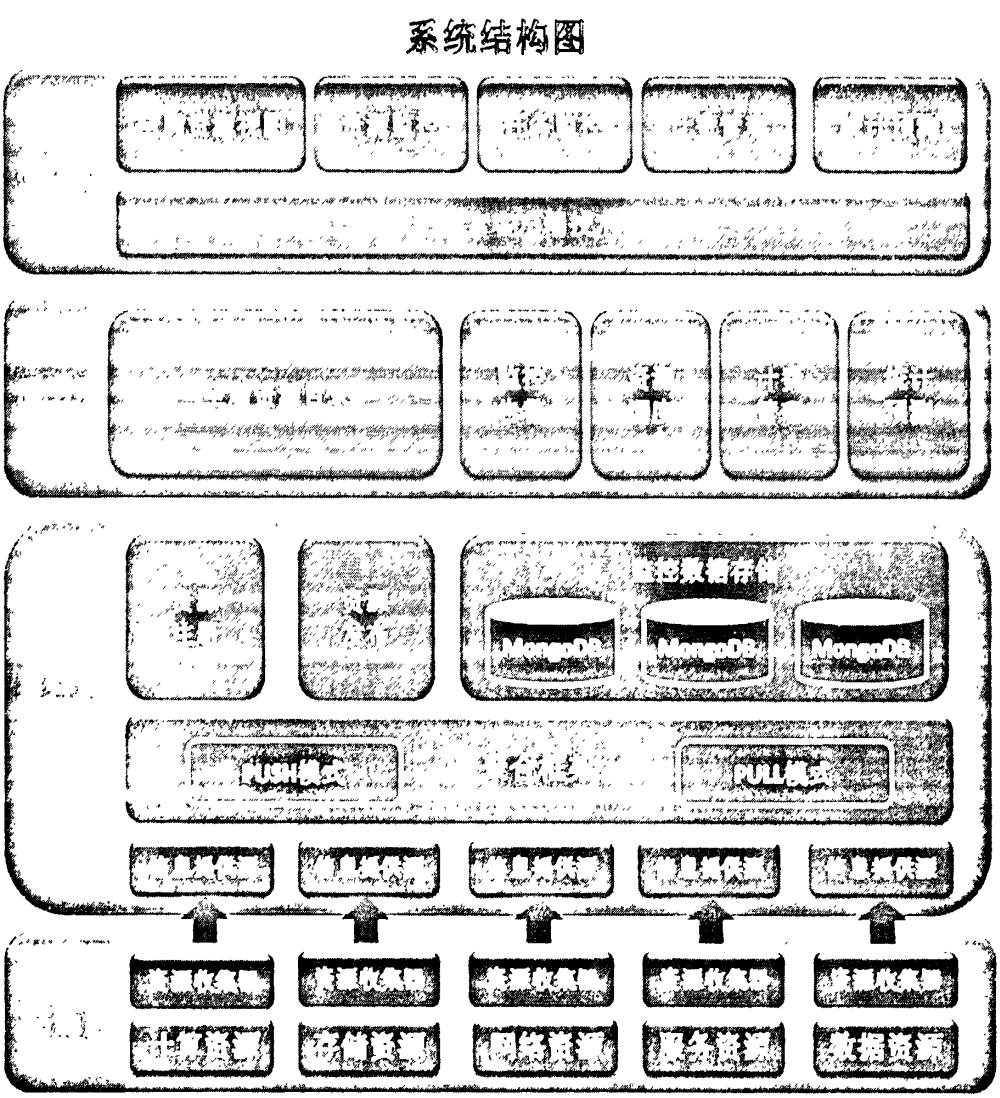

图 3.1 系统体系架构图

3.2.1 资源层

资源层是整个资源监控与管理系统获取和收集资源信息的来源。资源层为聚合层提供信息提供源（Information provider）。资源层获取和收集计算资源，存储资源，网络资源，服务资源，数据资源等各类资源的静态元数据信息以及动态资源状态信息，为整个资源监控系统提供输入数据。

资源层通过在各类硬件资源上部署监控代理，自动定期收集监控数据，并使用统一的资源描述方式，生产监控数据报告。通过聚合层和结合，采用推送和拉

去等方式，将资源监控数据传递给各个虚拟组织和最终用户，监控数据存储系统等。

3.2.2 聚合层

网格系统由大量分布式的异构资源组成，要监控的信息包括计算资源，存储资源，网络资源，服务资源，数据资源等各类资源的静态元数据信息以及动态资源状态信息，任务调度信息，系统日志等各种复杂多样的数据，需要对这些数据进行统一的描述和聚合。

聚合层的资源聚合框架提供 PUSH 和 PULL 两种数据聚合方式：

- PULL 模式：由聚合框架主动从资源层拉取数据。这种模式首先要为各类资源和数据提供注册，通过各类资源提交注册项中的更新周期等信息，定期去获取更新数据。这是聚合框架主要采用的方式。
- PUSH 模式：由资源直接向对其感兴趣的用户或者模块推送数据。用户可以订阅其感兴趣的资源和数据，当该资源数据状态发生变化时，更新数据会立即推送到用户手中。这种方式实时性更高，但是不适用于监控整个系统的资源数据。

聚合层基于资源聚合框架提供资源注册发布模块以及监控数据存储模块：

- 资源注册发布模块：为各类资源和数据提供注册和发布接口，对网格中的各类资源进行集中统一的管理，形成资源目录，为全局资源视图提供数据来源，支持资源的动态查询和资源发现。
- 监控数据存储模块：基于 MongoDB 实现海量监控数据的持久化存储服务，采用灵活的数据模型，支持高并发的读写，提供高性能可扩展的海量数据存储和访问，为服务层的监控告警模块、资源调度模块、统计分析模块、计费管理模块等提供丰富查询和访问接口。

3.2.3 服务层

服务层基于聚合层提供的数据以及接口，为整个网格监控系统提供服务。包

括：

- 全局资源视图：基于聚合层的注册信息，提供网格系统全局资源目录，资源访问方式，资源属性，生命周期等各种元数据信息，为资源发现及调度系统找到合适可用的资源提供判断依据。
- 监控告警模块：为展示层提供计算资源，存储资源，网络资源，服务资源的动态状态信息，让用户可以监控网格整体以及各部分的运行状态。并提供告警功能，当网格中的节点出现负载过高，资源不足等情况时，执行预设的相应脚本，并通过多种途径通知系统运维管理人员。
- 资源调度模块：通过对网格运行状态的把握，根据网格中各节点的负载情况，以及可用资源的多少，动态选择合适的资源来完成任务。资源调度模块，充分利用现有资源，尽快完成用户提交给网格的任务。
- 统计分析模块：通过对当前以及历史监控数据的统计和分析，评估网格的运行状态，提供网格整体以及各个节点的运行报表。并对网格运行状态的趋势进行预测，提前预报可能出现的系统运行瓶颈和故障，提高系统的稳定性和可用性。
- 计费管理模块：提供网格系统中各个虚拟组织对各类数据和资源的使用量， 通过对网格的资源进行计费管理，可以使得各个组织能加愿意提供自己的资源和数据，使得网格资源可以得到尽可能的充分利用。

3.2.4 展示层

展示层利用富客户端的可视化展示技术，从服务层获取数据并进行可视化展示。展示层根据不用用户的权限，为不同的用户提供不同粒度的监控信息。例如为网格的管理人员提供网格整体监控状态以及所以机器的详细状态信息，当网格系统中某些机器或者集群运行状态有异常时，为管理人员提供告警信息以及系统的日志；为普通的虚拟组织用户提供其可以访问的资源状态，负责情况等等。

3.3 本章小结

本章介绍了基于云架构的网格监控系统的整体架构设计。首先分析了网格监控系统应该满足的总体要求，以及应该实现的功能。接着给出了网格监控与管理系统的体系架构图，并分别介绍了资源层，聚合层，服务层，展示层各层的主要功能和作用。

第4章 基于MongoDB的海量监控数据存储模块

监控数据存储模块主要负责对网格监控系统获取到的各类网格监控状态信息进行持久化存储。由于网格系统包含大量软硬件资源，网格监控系统采集到的数据量大，更新频率快，如何高效的存储海量资源监控状态数据，提供灵活的数据存储模式，支持监控状态数据高并发的读写，提供易于扩展的存储能力，支持高效丰富的查询，保证数据的安全可靠是网格监控系统中的一个难点。

4.1 监控数据存储分析

网格监控系统采集的数据分为静态数据和动态数据。静态数据是是生命周期较长，相对静态的一些资源属性，比如计算资源的CPU个数，频率，存储资源的总容量等等。而动态资源则生命周期短，更新频率快，和系统运行状态相关联的资源状态信息，比如CPU的负责情况，CPU、内存、存储资源的使用量等。

对于静态数据更新频率慢，读的次数高于写的次数，对数据的安全性要求高。而动态数据刷新频率快，在机器规模大的集群上，并发写的请求非常大，而监控状态数据的安全性要求相对较低。所以考虑用适合各自特点的方式去存储各类监控数据。

通常监控数据可以采用集中存储或者是分散式存储两种方式。RRDtool (Round Robin Database Tool)是一个强大的数据存储工具以及绘图引擎，有很多监控软件采用RRDtool做监控数据的存储以及绘图。其中Round Robin是一种存储数据的方式，使用固定大小的空间来存储数据，并有一个指针指向最新数据的位置，形式类似于循环链表，随着新数据的写入，指针会随着数据的读写操作自动移动，新的数据会覆盖旧的数据。RRDtool使用这种方式来存储数据，并把数据存储在以“.rrd”后缀的文件中。

- 采用RRDtool来存储数据有一些缺点：

1) 一张性能图对应一个.rrd文件，小文件数量多。

2) 数据分别存储在部署gmeta数据收集器的机器上，并且分布在多个小文件中，监控数据是孤立的。如果要对集群的监控数据进行分析统计以及预测，需要访问大量机器上的大量小文件，对磁盘IO的压力很大。

3) 只能保留一段时间的数据，不能查看较长时间的历史数据。

4) 没有冗余备份，数据不够安全。

- 鉴于RRDtool的这些不足之处，我们提出了基于MongoDB的海量监控数据存储系统。我们采用 MongoDB 来存储监控资源状态数据，主要由如下原因：

1) 丰富灵活的数据模型：网格系统有动态性，异构性，分布性等特点，资源会动态的加入或者离开，经常会有新的资源添加进来，因此资源监控数据的存储也需要能够灵活的适应新加入的资源。传统的关系型数据库需要预先定义 schema，改动表结构会带来很大数据迁移的工作量。而MongoDB采用面向文档的方式存储数据，不需要定义模式，更加灵活方便。而且可以将文档或者数组内嵌进文档，一条记录就可以表示非常复杂的层次关系，易存储对象型的数据。

2) 易扩展性：随着互联网技术的发展，大规模集群使用的越来越普遍，而集群的规模也增长得非常迅速，对监控系统的可扩展性要求高，要能很好的适应所监控集群规模的迅速增加。对于传统的关系型数据库，通常采用分表的方式来扩展系统，分散负载。但是例如 MySQL 数据库并不支持自动分片，需要由应用系统来维护各个分表，手动进行分表，数据插入和查询是都需要由应用系统提供额外的分表处理逻辑。而MongoDB从最初设计就考虑到了扩展性的问题。它所采用的面向文档的数据模型，使其可以自动在多台服务器之间分割数据(auto-sharding)。它还可以很好平衡数据和负载，自动重排文档。这样需要更大的存储容量时，只要在集群中添加新机器就可以了。

3) 支持高并发的读写：监控数据量大，更新频率快。假设监控集群有 1000 个节点，每个节点有 50 个 metric 数据，数据每 15 秒更新一次，则每分

钟要写入 200000 个 metric 数据，因此资源监控状态数据存储需要支持高并发的写入。MongoDB 有着良好的读写性能。根据 MongoDB 的官方文档中介绍，当数据量达到 50GB 以上时，MongoDB 的访问速度是 MySQL 的 10 倍以上。

4) 支持通用辅助索引，可以进行多种快速查询：网格监控系统需要提供对网格系统各个资源监控状态的可视化显示，需要为资源调度和任务分配模块提供实时的资源负载状态信息，因此，资源监控数据存储系统要能够为不同的应用提供丰富高效的查询检索接口，才能满足各个模块对监控状态数据的需求。

4.2 总体设计

MongoDB 是一个面向文档的数据库，由 10gen 公司开发并维护，是非关系数据库中功能最丰富的，主要解决海量数据的访问效率问题。图 4.1 是基于 MongoDB 的海量监控数据存储系统的架构图。系统由分布式的多台服务器组成，我们采用了副本集与自动分片并用的架构，来构建 MongoDB 集群。

每个服务器上有多个分片，一个配置管理器和一个路由器。分片将集合切分成不同的片，将读写请求分散到集群中的不同机器，以达到系统负载均衡。配置管理器存储了元数据信息，包括哪些数据存储在哪个分片上。而路由进程从配置管理器读取数据和片的对应关系等元数据信息，当有读写请求发送给路由进程后，路由进程将请求转发到对应的分片，为上层应用隐藏了分片的细节。副本集通过对数据进行复制，保证数据的安全性和可靠性，自动应对故障切换，并可以提高读取性能。

当系统运行一段时间数据量增大，或者网格系统规模增大，需要更多存储空间时，可以通过添加新的服务器来达到扩展存储系统的目的。

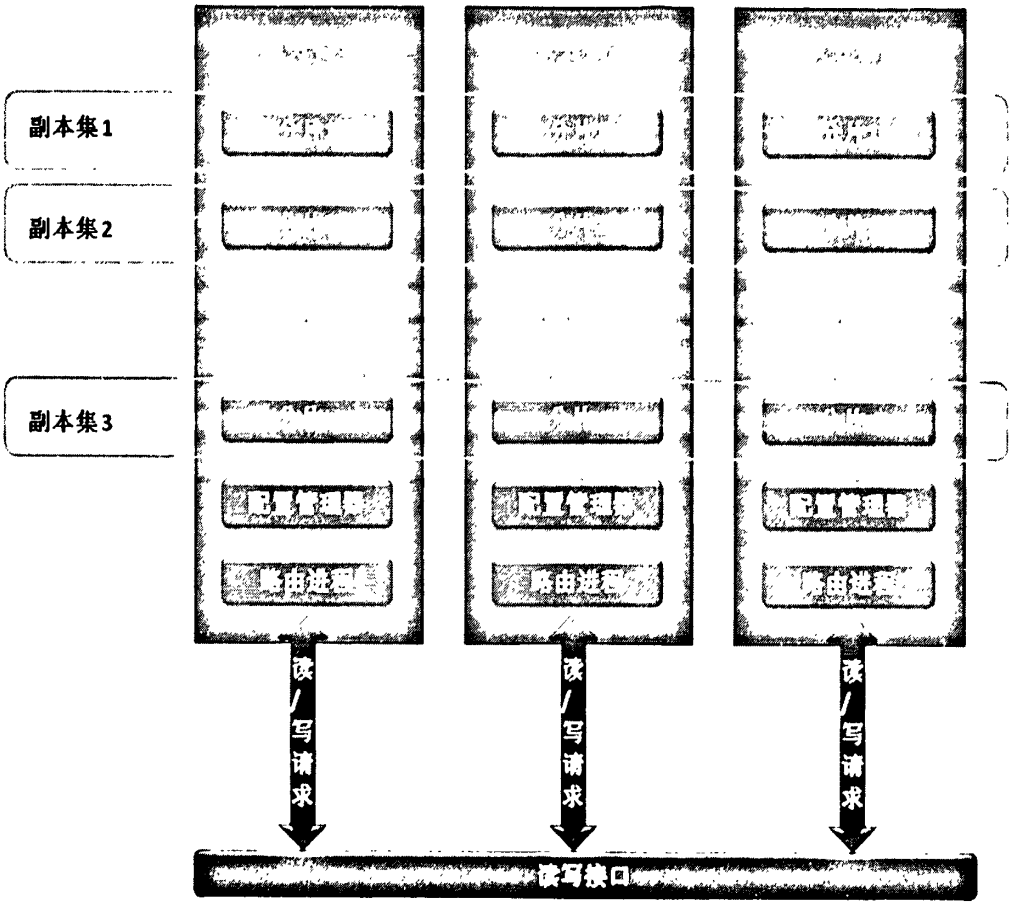

图 4.1 监控数据存储系统架构图

4.3 详细设计与优化

4.3.1 数据模型结构

数据模型对读写性能以及查询性能有非常大的影响，因此想要提供基于 MongoDB 的高性能的存储服务，需要有良好的数据模型设计。

文档是 MongoDB 中数据的基本单元，类似于数据库中的行的概念。集合可以包含多个文档，类似于数据库中的表。同样多个集合组成数据库。一个 MongoDB 实例可以承载多个数据库。我们把网格资源按照粒度划分为三个等级：grid（表示整个网格），cluster（表示网格下面的集群），host（表示集群下的机器）。我们为要监控的网格建立一个数据库，为每一个集群建立一个集合，每台机器每次收

集到的数据为一个文档，监控数据存储的层次结构如图 4.2。

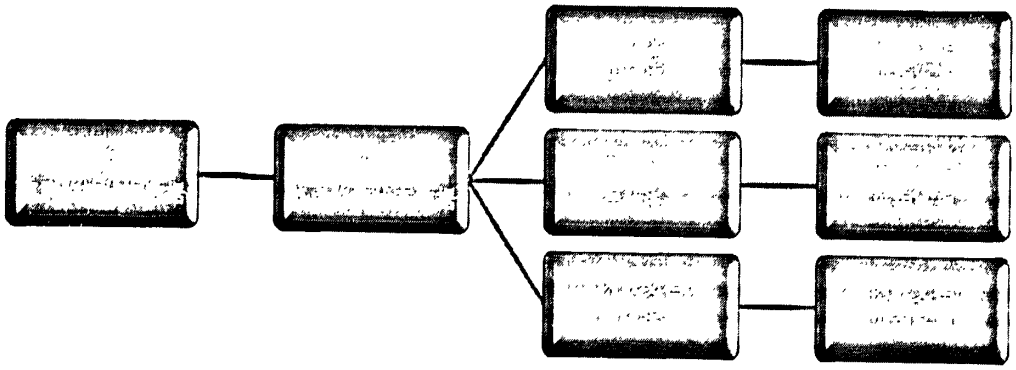

图 4.2 资源状态数据存储层次结构图

当我们从资源收集器获取到监控状态数据后，我们将该集群中机器的数据取出，构建文档后插入对应的集合，该文档包含对一个节点收集到的所有监控数据。文档结构如图 4.3。内部的蓝色框代表内嵌文档。通过内嵌文档的采用，很容易的表达了监控数据信息复制的结构关系。如果使用传统的关系型数据库，一个节点的监控数据首先需要拆分为静态信息表，计算资源信息表，存储资源信息表，网格资源信息表等。由于我们还提供了进程级的资源监控状态数据，例如计算资源的进程级占用情况，而由于进程数据有多条，即一个计算资源对应多条进程信息，所以还需要有对应的进程级计算资源信息表，进程级存储资源信息表，进程级网格资源信息表等。而当展示模块需要展示该资源的监控信息时，通常需要读取该节点各类资源的所有信息，这要通过读取多个表格做 join 操作来获取这些信息，而 join 操作是关系型数据库中最耗时的操作。而采用 MongoDB 的文档方式来存储，只要一条文档就可以将这些数据全部存储起来，而这些数据又通常是同时提取的，因此大大提高了读写的性能，提升了系统的效率。

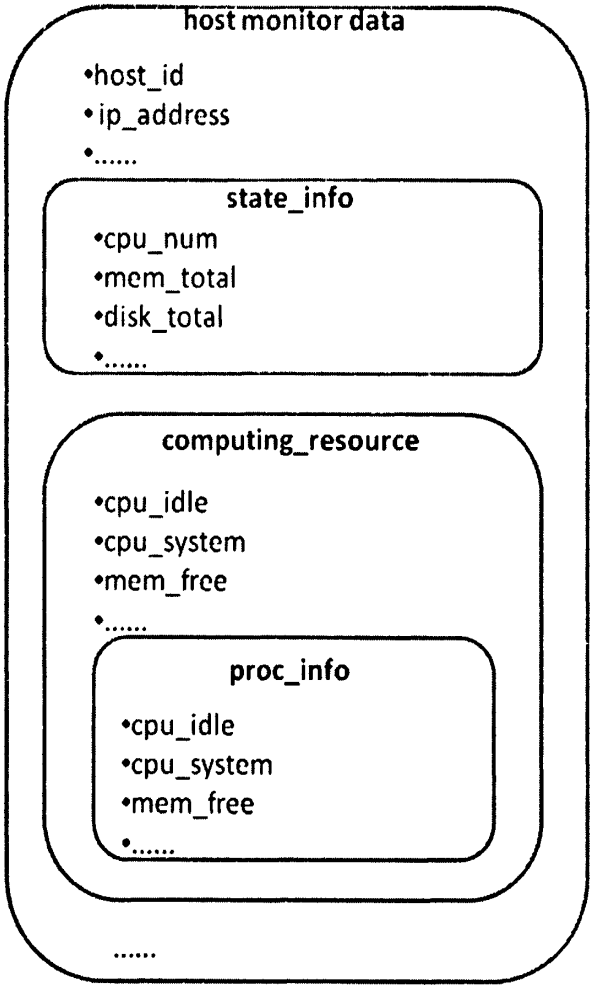

图 4.3 监控数据存储系统文档结构图

4.3.2 文档 ID 结构

MongoDB 中对于同一集合中的每一个文档，必须有一个_id 键来唯一标示这个文档。这个_id 键需要能够在分布式多进程的环境下保证唯一性，并且_id 很可能影响这文档的物理存储顺序。良好的设计_id 的生成方式，对于优化系统性能，充分利用数据的 locality 有着重要的作用。

MongoDB 中，如果创建文档时，没有给文档设置_id 键，则系统会自动生成一个_id 值赋给该文档。MongoDB 中自动生成_id 值为 12 字节，前 4 个字节为时间戳，接下来 3 个字节是所在主机的唯一标示符，接下来 2 个字符是产生该_id

值的进程标示符，最后 3 个字节是一个自动递增的技术器。

由于对资源监控数据进行读取时，通常会对相邻的机器进行读取，所以我们希望对于同一集群的机器的监控数据，同一台机器的监控数据物理上分布连续，而且按照时间顺序排列。所以我们设计了如图 4.4 的_id 键值。

图 4.4 文档 ID 结构图

其中前 3 个字节是集群 id 的散列值，接下来 3 个字节是主机名的散列值，中间 4 个字节是时间戳，接下来 2 个字节是 pid，以保证同一台机器上不同进程产生的文档 ID 唯一，最后 3 个字节是自增计数器，以保证同一进程产出的文档 ID 唯一。

4.3.3 查询接口

资源监控存储系统需要为系统其他模块提供丰富的查询接口。MongoDB 提供了丰富的查询功能，基本上可以满足传统关系型数据库对于单表的所有查询，因此我们采用基于 MongoDB 的监控数据存储系统，也可以为用户提供丰富的查询功能。例如：

- 给定资源 ID 和时间范围，获得对应节点最近时间内的监控状态数据
- 给定资源 ID 和要返回的元素数量，获得对应节点最近时间内的监控状态数据
- 查询符合某些条件（如 CPU 利用率小于 50%）的所有资源 ID。

4.3.4 聚合接口

网格监控系统通过对网格监控数据进行统计分析和预测，可以评估系统的运行状态，预测系统运行瓶颈等等。MongoDB 提供了很多强大的聚合工具，为网格监控数据的统计分析提供了很好的支持，例如：

- count：MongoDB中的count功能，可以快速统计出符合条件的文档个数。例如我们提供了统计集群正常运行机器数，统计集群平均负载等数据。
- group：MongoDB中的group功能，可以对选定的键进行分组。我们通过这个功能可以实现很多分组统计功能，比如计费管理中每个用户的资源使用情况，集群或者单台主机每天，每个月的各种监控状态数据统计等。
- MapReduce：我们通过使用MapReduce并行框架，可以高效快速的处理和分析海量监控状态数据，从海量数据中分析和挖掘有用的信息。MongoDB支持与MapReduce工具结合，可以实现很多复杂的统计预测功能

4.3.5 写入优化

- 插入方式选择

MongoDB提供多种插入文档的方式，包括BasicDBobject, BasicDBobjectBuilder, Map和json。我们通过对这几种插入方式进行了性能测试，选用Map的方式进行插入，测试结果及分析见4.4节。

- 批量插入

批量插入可以一次向集合插入多个文档组成的数组。批量一次发送数十个或者数百个或者更多的文档可以明显提高插入速度。因为对数据库进行一次插入，需要一个消息头告知数据库对那个指定的集合做插入操作，而一次批量插入只是单个TCP请求，可以减少处理大量重复的消息头的时间。另外磁盘对于写入操作需要进行寻址，把磁头调整到相应的位置，而批量写入可以减少磁盘寻址的次数，从而减少写入的时间。

因此，我们在设计资源收集器时，由资源收集器收集本组内的所有机器的监控状态数据，再一起通过聚合框架提交给监控数据存储模块，采用批量插入的方式来提高写入的效率。

- 自动分片

MongoDB支持自动分片，集群自动切分数据，分散到不同的机器上去，以达到集群的负载均衡，尤其是可以将写入操作分散到不同的机器上，提高写入的

效率。

MongoDB中有一个路由进程mongos，这个路由进程知道所有数据的存放位置，当应用需要读取数据时，它不需要知道数据具体在什么机器上，可以连接mongos来正常发送请求。而mongos并不永久存储数据和分片的对应关系，这些数据以及集群的配置信息，由配置服务器进行存储和管理，mongos只对这些数据进行缓存。

设置分片时，需要从集合中选取一个片键(shard key)，用该键的值作为数据拆分的依据。MongoDB进行自动分片时，会将选取的片键把数据划分为多个范围，并且尽量将不同的片分布在不同的机器上，这样对于新插入的数据，MongoDB会根据其片键的值在哪个范围内，选取对应的片进行插入。我们选取时间_id来作为自动分片的片键。由4.2.2节可知，_id设计为最前面是clusterid，这样选取_id作为片键，MongoDB会根据_id的范围进行分片，这样同一个cluster的数据会被放在一起。由于通常用户在请求监控数据时，同一个cluster的数据会同时请求，这样的设计可以很好的提高locality，提高读写效率。设置分片的代码如下：

```
grid.runCommand({"shardcollection" : "cluster", "key" : {"_id" : 1}})
```

4.3.6 读取优化

- 增加副本

通过对MongoDB增加副本，可以将读请求分布到各个数据副本节点上，以提高并发读取的性能。对MongoDB设置副本，还能够保证数据的安全性和可靠性，应对故障切换，作为离线批处理数据源等功能。

MongoDB提供多种复制模式，包括主从复制(master-slave)、副本集(replication set)等多种模式。

主从复制模式建立一个主节点和一个或者多个从节点，主节点有数据操作时，会记录下数据更改的日志oplog，从节点和主节点进行数据同步，通过主节点的oplog，采取和主节点相同的操作，以达到和主节点保持数据同步的效果。

副本集模式没有固定的主节点，整个集群会选举出一个活跃节点，其他的节点为备份节点。当该活跃节点发生故障不能正常工作时，集群中的其他节点会再次选举出一个活跃节点。选举活跃节点的工作，是根据每个节点的优先权以及最近一个同步时间等因素综合考虑决定的。因此副本集模式能够自动处理故障恢复。

- 设置缓存

设置缓存可以很好的提供高并发的读取请求，因为基于内存的缓存读取速度比硬盘要快很多倍，设置缓存可以分散数据库的压力，使得系统能够适应更大的规模的读写请求和数据。我们采用 LRU 和 Round Robin 两种替换策略，如图 4.5 所示。缓存按照 host_id 划分为多个槽，每个槽存放一台机器最近一段时间的监控信息。首先设定缓存的大小 m 和同时缓存机器数 n，这样可以控制缓存所占用的内存容量。我们的缓存系统最多同时缓存前面设定的 n 台机器的数据，当我们要访问新的机器时，采用 LRU 策略从缓存槽中找出最近最少使用的机器 id，将其数据清除。而每台机器数据缓存也有容量限制，我们采用 Round Robin 方式来替换数据，及采用类似于循环列表的方式，有一个指针指向当前可以写入数据的位置，当向缓存块写入数据时，该指针跟着写入数据的位置移动。当数据写到缓存块结尾时，将指针调整到存储块开头重新开始移动。

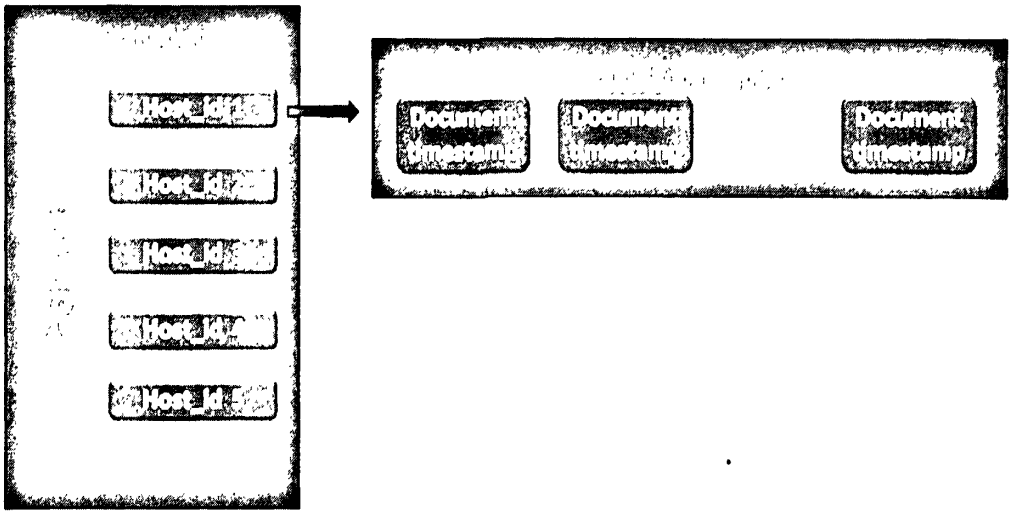

图 4.5 缓存系统结构图

4.3.7 查询优化

MongoDB 和传统的关系型数据库一样可以对任意字段创建索引。对查询常用的字段进行索引可以大大加快查询的速度，但是不能过分索引，因为构建索引也有一定的代价，会给数据写入带来额外的工作，比如每次插入、更新、删除时都会产生额外的开销，因此为合适的字段建立合适的索引非常重要。我们选择 host_id 键来建立索引，因为通常的查询是基于 host_id 来查找某个主机的监控状态数据的。

4.3.8 运行时备份

为了保证存储模块的可靠性和可用性，要对数据进行运行时备份。MongoDB 自带的 mongodump 工具，对运行的 MongoDB 做查询，然后把查询结果写入磁盘，就可以实现运行时备份。

4.4 性能测试及实验结果

4.4.1 测试环境

- 硬件环境

台式机 3 台

CPU：Intel Core Duo CPU E7400 @2.80GHZ

内存：4G

- 软件环境

MongoDB 版本：mongodb-linux-i686-1.8.4

MySQL 版本：mysql-5.1.60-linux-i686-glibc23

操作系统版本：Red Hat 4, kernel 2.6.9

4.4.2 MongoDB 写入性能测试

- MongoDB 不同插入方式的性能测试

- ✧ 测试方法：采用 BasicDBobject, BasicDBobjectBuilder, Map 和 Json 这 4 种方式向 MongoDB 数据库插入文档，每次插入数据 100000 条，测量插入数据所需的时间。我们一共进行了 8 组测试，用 8 组测试的时间和来衡量各种插入方式的性能。
- ✧ 测试结果：如图 4.6 所示，横坐标是不同的插入方式，纵坐标是时间，单位是 ms。从数据图中可以看出，采用 Map 方式进行插入所用时间最短，用 JSON 方式插入所用时间最长。
- ✧ 分析和结论：由于不同的插入方式构建数据模型的效率不同，所以插入的性能也会有所不同。由于 JSON 方式插入时，首先需要对 JSON 字符串进行解析，再构建对应的数据结构，所以消耗的时间比较长。我们采用 Map 的方式来构建文档并插入 MongoDB 数据库。

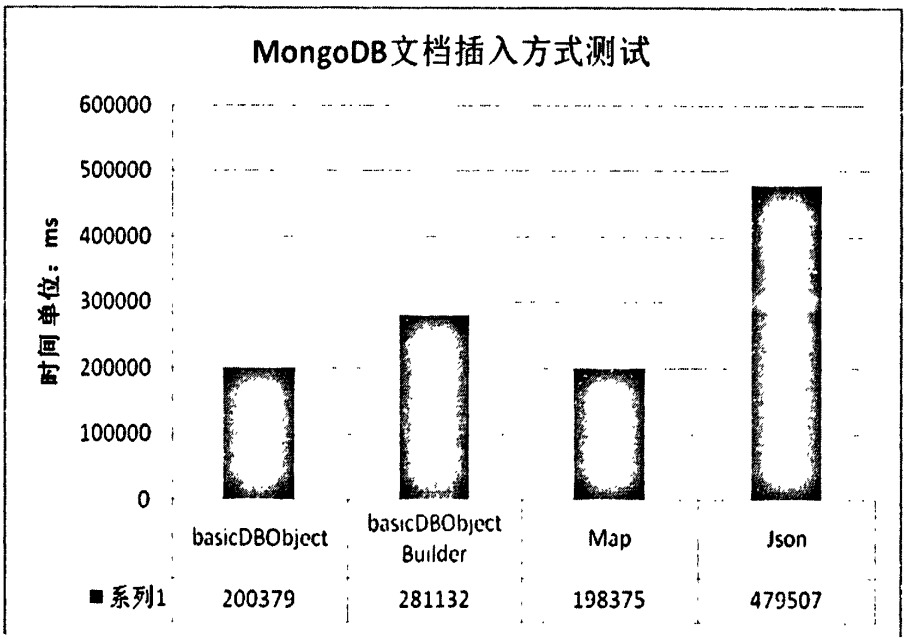

图 4.6 MongoDB 文档插入方式测试数据图

- MongoDB 并发写入性能测试
- ✧ 测试方法：通过循环创建了 100000 个线程，每个线程向数据库插入一条数据，测试每秒成功写入数据库的数据条数。
- ✧ 测试结果：每秒可以处理的数据量在 2000 至 3000 之间。如图 4.7，横坐

标是不同的秒数，纵坐标是每秒写入的数据条数。

✧ 分析与结论：MongoDB 写入对内存占用比较大，当内存逐渐写满后，插入性能下降。如果系统提高 MongoDB 数据库的写入性能，要尽可能的配置较大的内存。

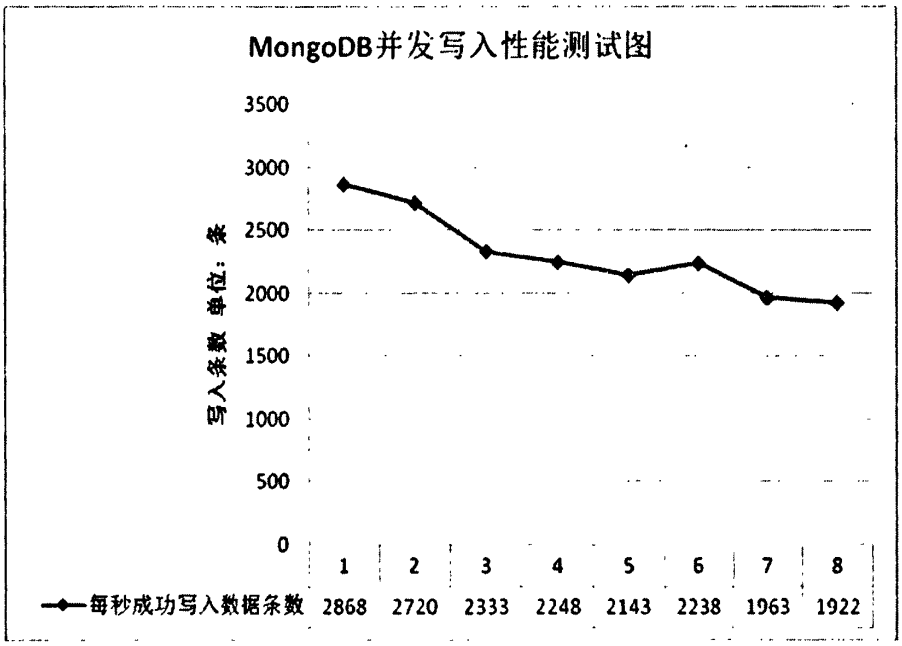

图 4.7 MongoDB 并发写入性能测试图

● MongoDB 安全写入与非安全写入性能测试

✧ 测试方法：向 MongoDB 数据库写入 100000 条数据，分片采用安全写入和非安全写入，测试使用时间，以及成功率。

✧ 测试结果：如图 4.8 所示，横坐标是测试的组数，纵坐标是写入数据所用的时间。由图可见，采用非安全写入的速度比安全写入的速度快大约 3-4 倍。其中非安全测试有失败 2 条数据。

✧ 分析与结论：安全写入需要对文档进行加锁，所以大大降低了写入的速度。在 5 组测试工写入 500000 条数据的情况写，失败了 2 条数据。这个对于金融交易等业务是不可接受的。但是由于监控系统的监控状态数据主要用于做统计分析与预测，对于这样的失败百分比完全可以接受。所

以我们采用非安全的写入方式，这样可以大大提高性能与效率。

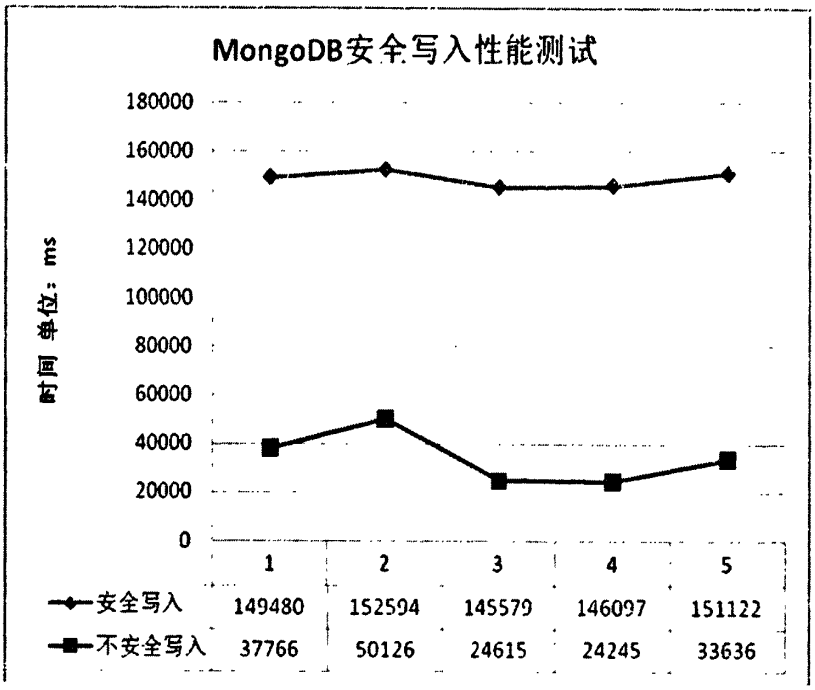

图 4.8 MongoDB 安全写入性能测试图

4.4.3 MongoDB 查询性能测试

- MongoDB 复合索引查询性能测试

✧ 测试方法：MongoDB 数据库中存储 1000000 条数据，对 host_id 和 timestamp 两个键建立索引，分别测量每个查询请求查询前 1 条记录和前 10 条记录，测试每秒钟可以处理的查询请求次数。

✧ 测试结果：如图 4.9 所示，横坐标是测试的秒数，纵坐标是 QPS（Query per second）。每个查询返回 1 条记录的 QPS 大致在 4.2k 左右，每个查询返回 10 条记录的 QPS 大致在 1.4k 左右。

✧ 测试结论：MongoDB 的对大数据量的查询结果不错。

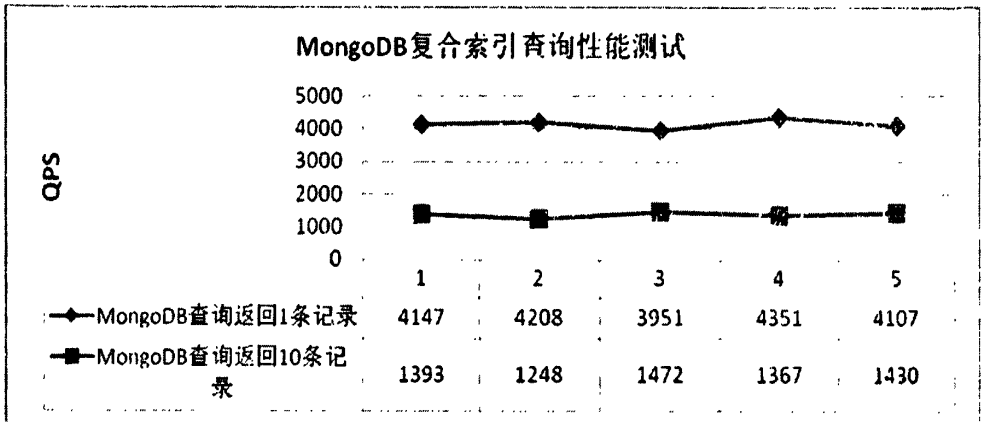

图 4.9 MongoDB 复合索引查询性能测试图

4.4.4 MongoDB 集群扩展性测试

- MongoDB 自动分片存储性能测试

✧ 测试方法：

1) 不设置分片，对 MongoDB 数据库连续插入 10000000 条数据，测试数据存储情况。

2) 设置自动分片，向 MongoDB 集群连续插入 10000000 条数据，测试 3 个分片测试数据存储分布情况。

✧ 测试结果：

1) 不设置分片时，数据存储在多个文件，文件大小分别为 64M，128M，256M，每次为前一个的 2 倍这样增长。

2) 设置自动分片时，数据基本均匀的分布在三个分片上，如图 4.10 所示。每个分片的文件存储方式和不分片时一样。

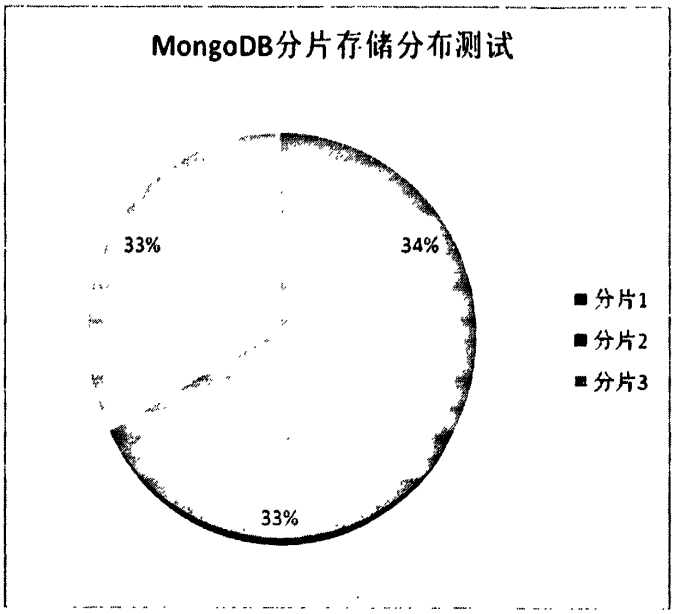

图 4.10 MongoDB 分片存储分布测试图

✧ 测试结论：由于测试数据比较整齐，每条数据的大小相同，所以三个分片上的数据非常均匀，可以很好的将分数分散到每个分片上。由于实际的监控数据，每条记录的内容类似，每条数据的大小也应该是基本相同的，所以分片能够分得比较均匀。

4.4.5 与 MySQL 数据库的性能比较

● 写入性能对比测试

✧ 测试方法：向 MongoDB 数据库和 MySQL 数据插入 1000000 条数据，测试每秒成功写入的数据条数。

✧ 测试结果：MongoDB 数据库的写入性能随着内存的消耗有所下降。MySQL 数据库的写入性能比较稳定。整理来看 MongoDB 数据库写入的性能明显要比 MySQL 数据库好很多。

✧ 分析与结论：MongoDB 数据库在写入时是先写在内存的 mmap 中，当内存中的数据量达到一定规模时，再一起写入硬盘，所以在内存充足时写入的速度非常快。但是当内存成为瓶颈时，写入的速度也就下降为磁盘

读写的速度了。MySQL 数据库支持事务，对数据的安全性更有保证，但是写入的性能比 MongoDB 差比较多。根据我们监控状态数据的特点，MongoDB 是很好的选择。

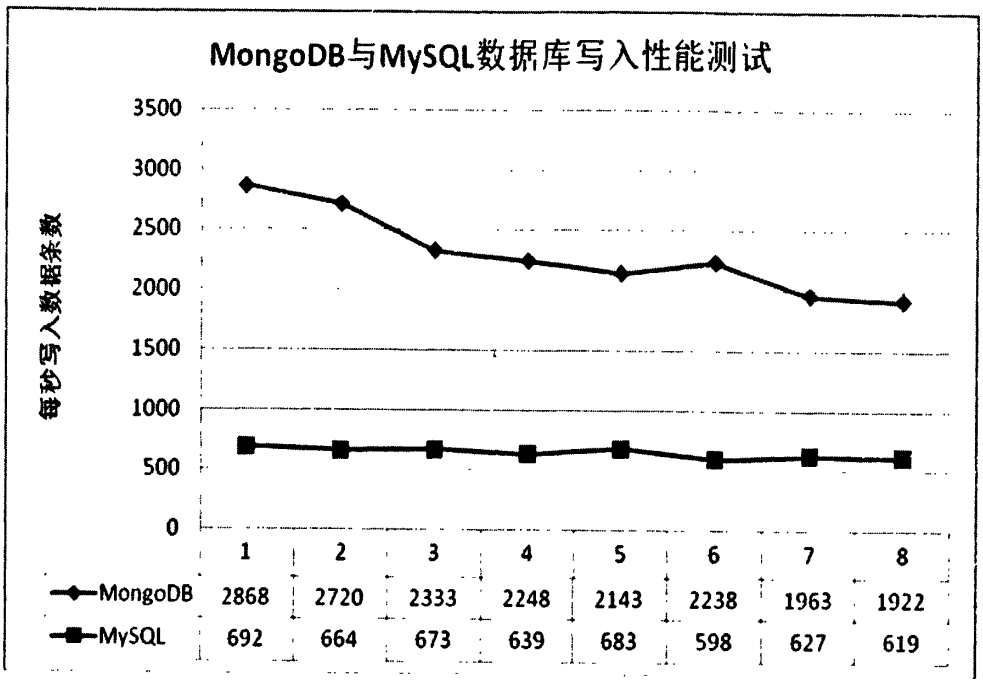

图 4.11 MongoDB 与 MySQL 数据库写入性能比较图

✧ 测试结论：MongoDB 设置了自动分片，并且选择正确的片键，可以很好的将分数分散到每个分片上。

● 查询性能对比测试

✧ 测试方法：MongoDB 数据库和 MySQL 数据库中都存入 1000000 条数据，对 host_id 和 timestamp 两个键建立索引，分别测量每个查询请求查询前 1 条记录和前 10 条记录，测试每秒钟可以处理的查询请求次数。

✧ 测试结果：如图 4.12 所示，横坐标是测试的秒数，纵坐标是 QPS（Query per second）。

✧ 测试结论：在单表复合索引的情况下，MongoDB 的查询性能比 MySQL 数据库要好。

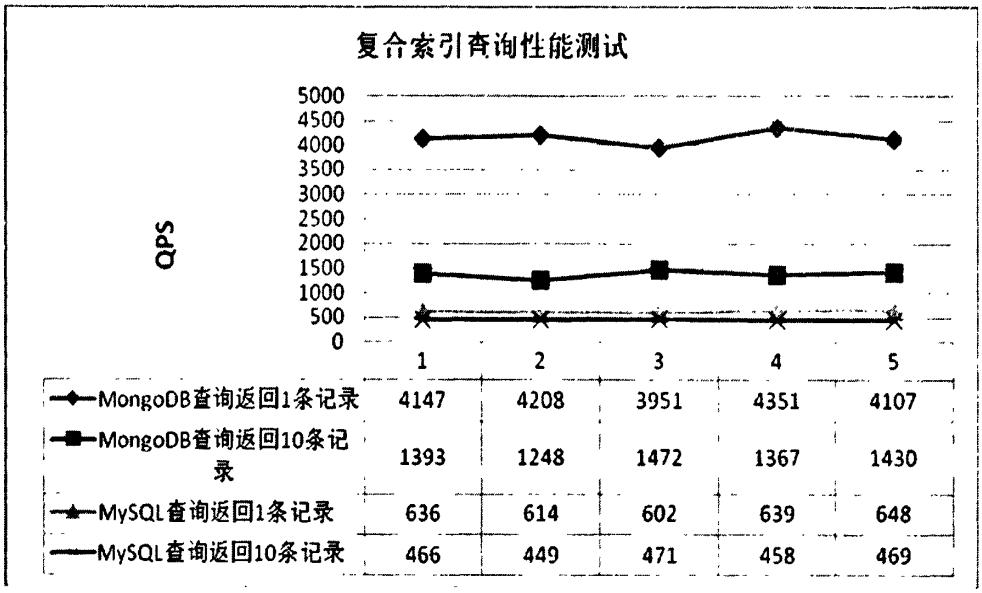

图 4.12 复合索引查询性能对比测试图

4.5 本章小结

本章首先简要对资源监控数据的特点进行了分析，列举了采用 MongoDB 数据库存储资源监控数据的优势。接着给出了基于 MongoDB 的海量监控数据存储模块的总体设计，又从多个方面介绍了该模块的详细设计与优化方法。最后对 MongoDB 进行了一些性能测试和分析。

第5章 基于一致性哈希的高效易扩展资源聚合模块

网格系统包含海量分布式异构资源，网格监控系统需要高效的将这些资源的监控状态数据进行聚合，把数据提供给不同的虚拟组织和用户，以及监控数据存储系统。如何使得资源聚合框架能够实时、高效、易于扩展是网格监控系统中的一个难点。

5.1 一致性哈希算法研究

一致性哈希（Consistent Hashing ）[17]是麻省理工学院于1997年提出的一个分布式哈希表（Distributed Hash Table，DHT）的模型，用于解决在动态的网络中，增加或者删除一个节点不会显著的改变已有的键值映射。一致性哈希模式被广泛应用在解决分布存储和路由问题上。

- 一致性哈希函数应该满足以下条件：

1) 平衡性（Balance）：指的是能够保证哈希的结果，尽可能均匀的分散到各个机器上，侧重于访问请求的平衡。

2) 伸展性（Spread）：在分布式环境中，用户不一定能够看到所有的机器，每个用户可能只能看到一部分，这样相同的内容被不同用户映射后存储在不同的位置，这样就会降低系统存储的效率。而一致性哈希应当使得相同的内容存储在同一位置，可以降低系统冗余度。

3) 单调性（Monotonicity）：当有新的机器增加或者有已有机器减少时，需要对数据进行重新哈希来重新分配数据和机器的对应关系。一致性哈希应当能够保证尽量少的移动原有已分配的内容，即已分配的内容只能从已有的机器移动到新加入的机器上，而不能移动到旧的机器上。

4) 负载性（Load）：负载分散，和平衡性类似，更侧重于数据存储的均衡。

一致性哈希只描述了一种模式，它有多种实现算法，关键在于如何定义数据分割策略和节点快速查询策略。下面介绍一种典型的对一致性哈希算法的实现。

- Chord 算法

Chord[18,19]是一个 P2P 环境下的分布式哈希表的协议和算法。Chord 给出了 key 是如何分配到节点，以及一个节点如何根据给定的 key 值找到对应的 value。使用 Chord 查找协议，节点的 key 值被分配到一个圆环上，这个圆环最多有 2^m 个节点。节点和 key 值通过一致性哈希分配到一个 m 位的标示符。Chord 采用 SHA-1 算法作为哈希函数，SHA-1 会产生一个 2^{160} 的空间，每项为一个 16 位的大整数。我们可以认为这些整数首尾相连形成一个环，称之为 Chord 环。整数在 Chord 环上按大小顺时针排列，Node（机器的 IP 地址和 Port）与 Key（资源标识）都被哈希到 Chord 环上。

分布在 Chord 环上的节点理论上是随机分布的。但是在节点数比较少的情况下，分布肯定是不均匀的，所以为了保证 Chord 算法的均衡性，采用增加虚拟节点的方法来增加其平衡性。

假设整个网络有 N 个节点，当计算出 key 的哈希值，需要找到离其距离最近的节点时，最简单的方式是顺序查找，这样的时间复杂度是 O(N)，但是当节点数很多，节点经常的加入和推出的 P2P 网络来说，效率是非常低的。因此 Chord 提出了一个非线性的查找算法来加快查找的过程：

1) 每个节点都维护一个路由表（Finger 表），该表长度为 $\log_2 N$，该表的第 i 项存放该节点的第 2i 个后继($1<=i<=\log_2 N$)。
2) 每个节点都维护一个前驱和后继列表，该列表的作用是能快速定位前驱和后继，并能周期性检测前驱和后继的健康状态。
3) 存储：沿 Chord 环，hash(Node)>=hash(key)的第一个 Node，我们称这个 Node 为这个 Key 的后继。同时为了保证数据的可靠性，会顺时针往下找 K 个冗余节点，存储这个数据。
4) 查询：给定一个 Key 值，先从自己的路由表中，找一个和数据 id 距离最近、并且存活在网络中的节点后继。如果该节点的 id 巧合和数据 id 相等，那么找到该 Key 值的存储位置。如果不相等，则对应路由表进行递归查找。一般或需要经过多次查询才能找到数据所在的节点，如图 5.1

展示了从 N0 节点查找到 N21 节点的过程。查找的次数是可以被证明小于等于 $\log_2 N$ 的。

5) 新增节点：需要知道网络中存活的一个节点 j，然后通过和节点 j 交互，更新自己和其他节点的路由表，并将离自己距离最近的节点中的数据进行复制，以提供数据服务。

6) 移除节点：路由算法会自动跳过这个节点，并且依靠数据的冗余来持续提供服务。

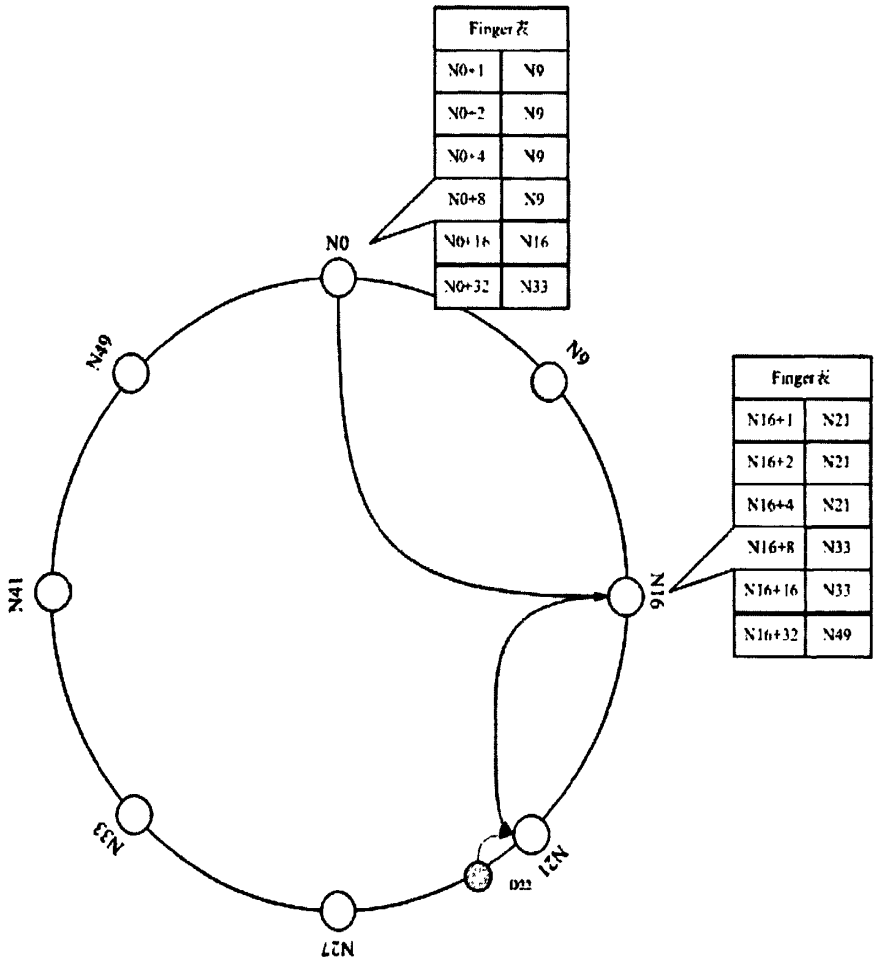

图 5.1 Chord 算法的查找过程示意图

5.2 现有资源聚合框架存在的问题

监控与发现服务(Monitoring and Discovery Service，MDS)是 Globus 提供信息

服务一个模块，是资源聚合技术的主要框架，它提供资源的注册和发布功能，并能够对资源属性进行查询，来发现合适的资源。Gloubs toolkit 4 中的 MDS4 是基于 WSRF（Web Services Resource Framework）框架实现的，包括信息提供者(Information Provider)、索引服务(Index Service)、触发服务(Trigger Service)、监控可视化(WebMDS)等组成部分。

其中的索引服务负责注册以及数据的缓存，由于 MDS 不提供持久化的存储，所以向 MDS 查询时，是查询索引服务缓存的数据。如图 5.2 所示，标注 R 的圆圈代表资源，标注 A 的圆圈代表索引服务，细箭头代表资源监控数据注册的流向。最顶层的索引服务可以聚合下面 6 个资源提交的所有信息，但是注册的层数越多，更新周期就时间越长。如图中资源每 30 秒钟更新一次数据，到第一层索引服务每 5 分钟更新一次数据，而最顶层的索引服务每 10 分钟更新一次数据。因此索引服务只能保证数据是最近的，但是不能保证数据是最新的。用户想要获得最新的数据需要直接和资源进行交互。

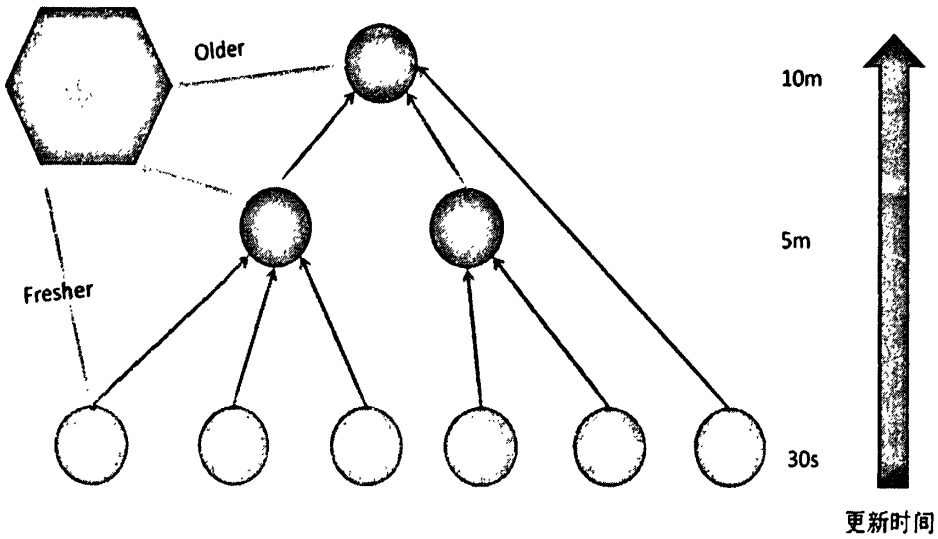

图 5.2 MDS 索引服务层次图

- 由此，MDS 用于资源聚合存在的问题主要有：

1) 索引服务采用树状的层次结构，配置较为复杂，比较难实现动态的负载均衡，系统的可扩展性查。

2) 如果索引服务的一个节点失效，则注册到该索引服务的资源数据就都会丢失，造成单点故障。

3) 当系统规模非常大时，需要多层树状索引来降低单台机器的负载。这样资源监控数据需要经过多个索引服务节点，会降低获取到的数据的实时性，

4) MDS 基于 WSRF 框架，使用 XML 文档来传输数据，冗余的标签信息多，造成传输数据相对低效。

5.3 总体设计

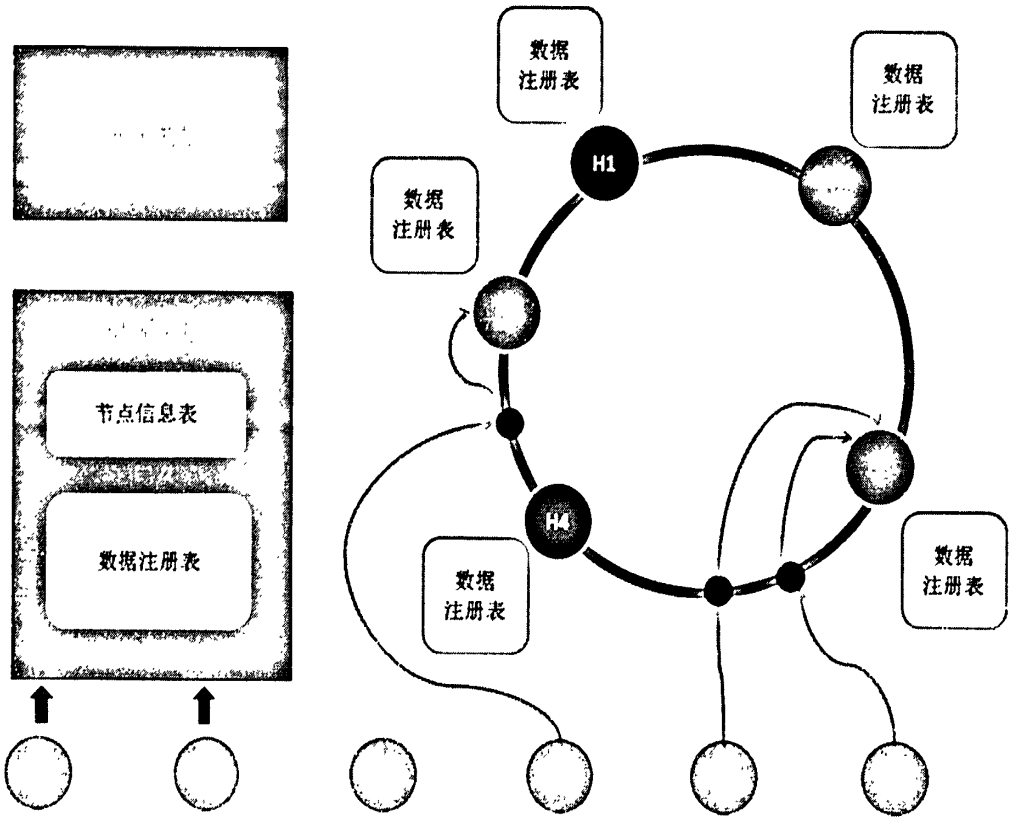

图 5.3 资源聚合模块架构图

为了解决已有资源聚合框架存在的问题，我们提出了基于一致性哈希的 P2P 结构来代替 MDS 中索引服务的树形结构，该方式可以自动选择和配置资源应该注册到哪个索引服务，易于实现负载均衡，易于扩展，自动应对单点故障。图 5.3

为基于一致性哈希的高效易扩展资源聚合模块的系统架构图。

其中H为索引服务器，通过一致性哈希算法分布在一个圆环上。R为资源，资源首先注册在注册管理器中，再根据一致性哈希算法计算出应该注册在哪台索引服务器上。如图中R4分布在H5上，R5和R6分布在H3上。注册管理器负责维护和管理节点信息表以及数据注册表。节点信息表维护了所有索引服务器的信息，数据注册表维护了资源和索引服务器的对应关系。数据的注册由资源发起，由注册管理器来进行自动分配和注册。每台索引服务器本地也维护一个自己的数据注册表，索引服务根据这个表里注册的内容，定期向资源监控服务pull数据。查询管理器负责将用户的查询请求转发给对应的索引服务器，并将索引服务器返回的结果进行集成和处理，再返回给用户。

5.4 详细设计与实现

5.4.1 Hash算法与Key值的选取

使用一致性哈希算法首先要设计一个哈希函数，我们采用MD5（Message Digest Algorithm 5）摘要算法来做哈希函数。MD5信息摘要算法是被广泛使用的散列函数，可以用任意长度的信息生成一个128位的信息摘要。

我们采用ip地址和服务端口号作为哈希函数的key值，配置管理器获得主机的ip地址和服务端口号，并根据一致性哈希方法计算，可以得出该主机的哈希值，用于判断该主机应该注册在哪台索引服务器上。

5.4.2 负载均衡

负载均衡是要做到根据索引服务器的每个节点其机器性能的不同，使得资源监控数据尽量均匀的注册到每个索引服务器上。

按照前一节的hash算法，索引服务器会被分配到$0\sim2^{128}-1$的范围内。当节点数量较少时，容易造成节点在环上分布不均匀，而且每台机器的性能配置不一样，应该承担的负载也应当有所不同。为了达到负载均衡的目的，我们引入了虚拟节

点来解决这个问题，将一个物理节点映射为多个虚拟节点。

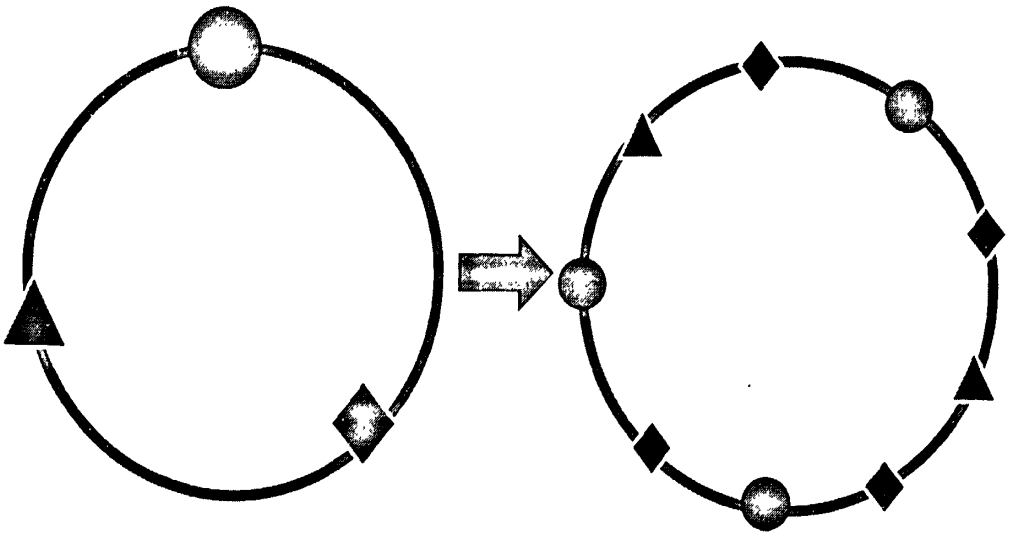

图 5.4 支持虚拟节点的一致性哈希算法

如图 5.4 所示，左图的三个物理节点，映射为右图的虚拟节点。相同形状表示同一个物理节点对应的虚拟节点。三角形节点性能较差，映射为 2 个虚拟节点，菱形节点性能强，映射为 4 个虚拟节点。这样做有如下优点：

1) 性能强大的物理机器映射为多个虚拟节点，性能较差的物理节点映射相对较少的物理节点，这样就可以隐藏不同物理机器性能不同的问题，将各个虚拟节点看成性能相似的节点。
2) 当新加入物理节点后，该节点对应的虚拟节点分散在环上的不同位置，可以分担其他多个节点的负载。
3) 当某个物理节点出现故障时，同理，该节点的负载也可以分担到多台其他的机器上，避免都分配给其相邻节点，使得相邻节点的负载突然增大。

5.4.3 注册管理器

注册管理器管理和维护资源的注册项以及各个索引服务器的数据注册表。图 5.5 展示了资源向注册管理器注册的一系列操作，操作步骤如下：

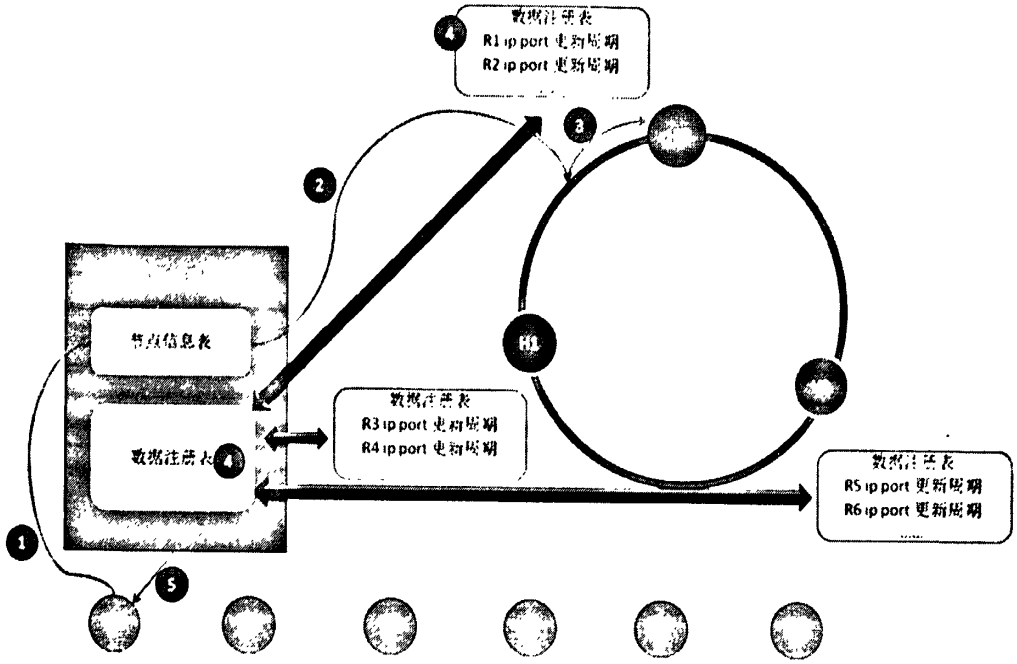

图 5.5 注册管理器资源注册示意图

1) 所有需要被监控的资源R1首先要向注册管理器注册，注册项包括ip地址，本节点上监控收集服务的端口号，所要注册的索引服务名称，数据更新周期等。

2) 注册管理器收到该注册项后，根据 ip 地址和端口号，计算其 MD5 值，得到在一致性哈希环上的位置。

3) 注册管理器维护一个索引服务器的节点信息列表，包括每个索引服务器的 ip 地址，哈希值等，并且按照在一致性哈希环上的位置大小进行排列。注册管理器通过二分查找发，可以快速的找到一个距离该节点最近的索引服务器，这里即 H2.

4) 更新索引服务器上以及注册管理器上的数据注册表。每个索引服务器有一个数据注册表，这个注册表内有所以注册到该服务器的资源的IP地址，服务端口，更新周期等注册项信息。索引服务器根据数据注册表定期去拉资源的监控数据。同时，注册管理器本地维护了所有索引服务器的数据注册表，当某台机器出现故障时，可以方便的知道之前是哪些资源注

册在该台机器上。

5) 将选中的索引服务器地址，以及索引服务的端口号返回给注册资源 R1。这样做是为了支持推送的方式，当 R1 的数据发送变化时，可以推送给对应的索引服务器。

5.4.4 分布式查询与集成

当用户需要对资源数据进行查询时，需要有一个机制能够将查询分配到对应的索引服务器上，并查询到的结果进行集成。常见的查询有一下三种情况：

- 当用户给定资源主机 id，查询该资源的相关数据时。首先计算出该台主机的哈希值，再根据注册管理器的节点信息表，找到该主机的信息注册在哪台索引服务器上。进而对该索引服务器上的索引服务进行查询，并将结果返回。
- 当用户给定其他条件，例如查询所有硬盘容量大于 50%的机器时，首先到每个索引服务器进行该条件的查询，然后返回所有机器的查询结果的集合。
- 当用户要求对某个条件排序时，例如查询 CPU 占用率最高的 50 台机器。首先对每个索引服务器进行该条件的查询，对所有机器的查询结果做归并排序，再取出前 50 个查询结果返回。

5.4.5 故障自动检测与处理

- 故障自动检测

索引服务器会定期向注册管理器发送 heartbeat，同步数据注册表。当注册管理器发现连续几个周期没有收到某台机器的 heartbeat 时，则认为该机器出现了故障。

- 故障自动处理

当注册管理器发现一台索引服务器出现故障后，会将该机器的信息从节点信息表中移除。并将该节点的数据注册表中的所有数据，重新注册到现有的机器上，

同时更新对应索引服务器上的数据注册表，以及注册管理器本地的数据注册表。

5.5 本章小结

本章首先简要介绍了一致性哈希算法以及对一致性哈希算法的一种实现。然后分析了现有资源聚合框架 MDS 存在的问题。根据 MDS 的不足，我们给出了基于一致性哈希的高效易扩展资源聚合模块的总体设计，并详细介绍了该模块的具体设计与实现。

第6章 基于云架构的网格监控系统的详细设计与实现

6.1 资源层的设计与实现

资源层主要负责获取和收集各类资源监控数据。其中资源状态采集模块是资源层的主要模块，主要负责从网格集群各个节点上收集各种监控状态数据，并提交给聚合层。

6.1.1 监控对象及事件

网格环境下资源内容非常丰富，结构复杂，包括计算资源，存储资源，网络资源，服务资源等多种资源。监控系统获取的数据主要分为静态信息和动态信息两种

1) 静态属性信息。静态信息主要是各种软硬件资源的元数据信息，用于描述资源的属性信息，如共有的资源名称、描述，计算资源的CPU个数，内存容量等，存储资源的硬盘容量，文件系统类型等，网络资源的带宽，以及服务资源的服务名称、访问地址等。这些静态属性数据更新频率低，数据量相对较小。

2) 动态资源状态信息。动态资源状态信息主要是各种软硬件资源的状态数据信息，用于描述资源随时间变化，资源状态的变化，如CPU的使用率，内存的使用率，负责变化等，还包括进程级别的各种资源使用状态，比如使用计算资源的进程的名称、CPU使用量、内存使用量、进程ID等等。由于随时间不停变化，所以这些数据更新频率高，数据量大。

- 监控事件是与监控对象紧密联系的，通常是在特定的事件的输出结果，包括：

1) 警告：当主机的负载超过限制，或者CPU及内存使用率超过限制，系统运行存在着某种隐患时，应当及时以适当的方式进行告警，使得运维管理人员能够知道系统运行可能出现的问题，及时加以解决，防止对系统运行造

成致命的威胁。

2) 错误或异常：当系统运行用户提交的作业时，可能会出现一些错误或者异常，比如空指针(NullPointException)，数组越界等。监控系统应该能够将这些错误或者异常信息，详细的记录在系统日志中，这样可以帮助运维人员定位错误的位置。

3) 故障：当系统中的某些机器出现故障不能正常运行，如物理存储器的可用空间为零，或网络流量过大发生阻塞等情况时，监控系统应该能够及时发现这些出现故障的机器，并通知给运维人员，及时采取相应的措施，以保证系统平稳的运行。

6.1.2 资源采集模块的设计与实现

资源状态采集模块主要负责从网格集群各个节点上获取并收集各种监控状态数据，并提交给聚合层。在每台被监控的机器上，都会运行一个数据采集器，负责获取该节点机器的状态信息；在每个主节点上，都运行一个数据收集器，负责将集群中其他节点的数据收集起来，提交给聚合层。资源状态采集模块的体系架构图如下：

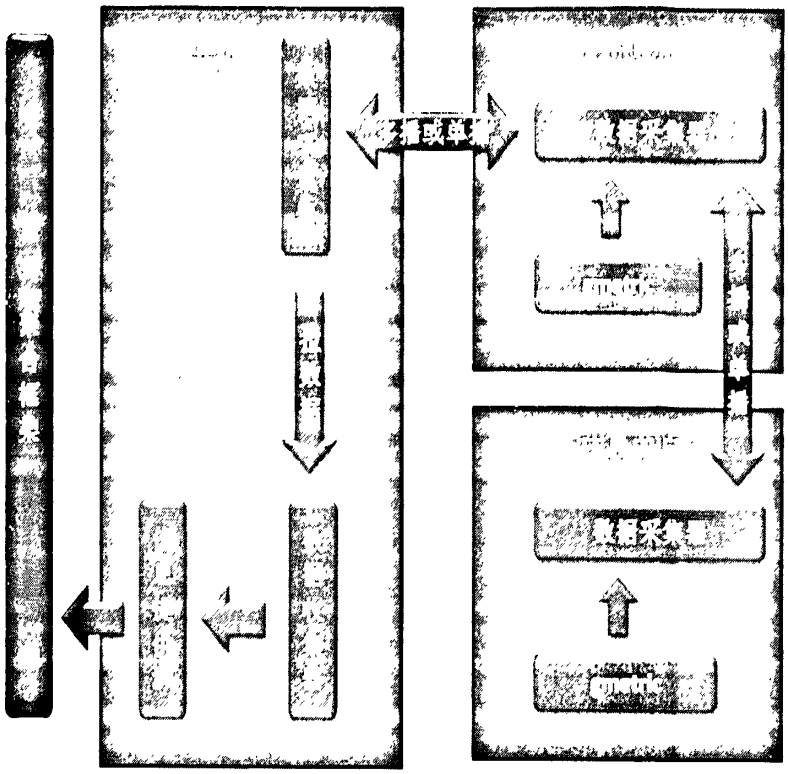

图 6.1 资源采集模块结构图

6.1.2.1 数据采集器

数据采集器安装在需要监控的所有机器上，它是运行在被监控机器上的多线程守护进程，将从操作系统和指定主机中收集资源的状态信息，用多播或者单播的方式向集群中的其他机器发送监控数据，并向数据收集器提供监控数据。数据采集器要支持 Linux，Unix，Windows 等常见的操作系统。其每个线程的具体任务如下：

1) 收集和发布线程用于采集节点的 metrics 并组播出去；收集基本的数据，如系统负载，CPU 利用率，内存利用率等。它同样可以发送用户使用 c 或 python 定义扩展模块所采集到的其他监控数据。

2) 监听线程用于监听组播端口，并把这些 metrics 保存于内存中的一个多级 hash 表；

3) 一组 XML 导出线程组用于响应 TCP 请求，把簇内的 metrics 导出成 XML 文

档格式，并发送出去。

数据采集器不会持久化保存数据，仅仅是将监听到的数据保存在内存中，并相应发送数据。节点间通过心跳信号检测对方节点存活与否，如果一段时间内该节点没有广播 metrics，我们视其为宕机，而且每次启动时，会广播一个数据采集器的启动时间，这时邻居节点收到以后就视其机器重启，会删除该节点已存的所有 metrics。

Metrics 数据由数据采集器内置的程序或 gmetric 程序获得，一般以 XDR 形式压缩保存，保存格式为：(key，value)，其中 key 为 4 字节，value 为 4-8 字节。metrics 的采集次数、频率和发送时间间隔均在配置文件中定义。数据采集器维持一个采集表，每个 metric 都有其属性。

Ganglia 是由加州大学伯克利分校开发的一个可扩展的分布式监控系统，用于网格环境下的计算资源、存储资源、网格资源和服务资源的监控。Gmond(Ganglia Monitoring Daemon)是 Ganglia 中负责收集和发送度量数据（如处理器速度、内存使用量等）的守护进程。Gmond 消耗的系统资源非常少，几乎不能影响机器的性能，因此我们采用 Ganglia 作为数据采集器。

资源采集器和收集器安装完后，需要设置一些配置项才能使其正常工作。Gmond 的配置文件的位置为/etc/ganglia/gmond.conf。Gmond 配置文件中共有 9 个配置段，具体作用介绍如下表所示：

表 6.1 Ganglia-monitor 配置说明表

Sections	Description
cluster	这一段是对集群相关信息的配置。其中 name 字段定义了节点所在的集群信息，所有 name 相同的节点将认为在同一个集群里，其他变量是对该集群的说明。
host	只有一个变量，location，是对这个节点的一些说明，可根据需要配置。

Globals	gmond 的全局配置，一般不需要修改。
udp_send_channel	Ganglia 提供了灵活的监控数据传输方式，可以采用多播或者单播的方式。这一部分定义了 upd 发送的端口信息，可以定义多个这样的段。ganglia 默认采用多播发送监控数据，其中的 mcast_join 变量指定多播组。但是，并非所有环境都适合多播，因此 ganglia 也支持单播的方式，host 变量指定一台 gmond 服务器接受监控数据。值得注意的是，mcast_join 和 host 不能同时出现在一个 channel 里。port 指定端口号，ttl 指定跳数一般为 1，如果需要经过 gmond 中转，需要修改这个跳数。
udp_recv_channel	这一部分对应于 udp_send_channel 段，是对 udp 接受端口信息的配置，也可以设置多个。如果采用单播的方式，需要注意 bind 变量指出的 ip，你的所有其他 gmond 可以访问到，事实上，可以使用 0.0.0.0。
tcp_accept_channel	指定一个端口，可以通过 TCP 的的方式，从这个端口读取 XML 格式的监控数据。
collection_group modules include	这三个段用于配置具体的监控项、监控间隔等信息，也可以配置自定义的监控项。

6.1.2.2 数据收集器

数据收集器运行在集群的主节点上，主要负责周期性的向集群中的数据采集器发送轮询包，收集监控数据，并将收集到的数据提交到上层聚合层。一个数据收集器可以收集一个或者多个集群中的信息，一个集群中也可以有一个或者多个数据收集器。

gmetad（Ganglia METAdata Daemon）是Ganglia中负责定期收集所有监控数据项并存储在RRD存储引擎内的守护进程。Gmetad将从data source采集到的metrics，经由SAX XML进行解析，存储在内置一个gperf的hash表中，便于数据的处理，最后将处理好的数据存于RRDTools中。Gmetad的配置文件位置为/etc/ganglia/gmetad.conf。其中data_source是最重要的参数，表示了收集器收集哪个分组的数据。Data_source的值应该对应于Gmond中的cluster name的值。如果要收集多个分组的数据，可以配置多个data_source字段。通过配置Ganglia可以实现分组监控。所谓分组监控，是指让同一个集群或者同一个节点的机器通过多播的形式汇集到一起，网格监控服务器只需要访问一台机器，即可获得这个集群或者节点的所有机器的信息。这是Ganglia实现超大规模集群监控所依赖的主要机制。

由于ganglia的监控数据是孤立的，是分别存储在部署在gmeta的机器上，并且不同的数据存储在不同的rrd文件中的，io性能差，不利于对大量监控数据进行分析。所以我们采用gmetad的数据收集功能，但不使用RRDTools中的存储和可视化展示功能。详细的存储和可视化方法见第五章和第六章。

6.1.2.3 添加额外的监控项

由于网格资源的多样性以及异构性，我们的资源状态采集模块需要有良好的可扩展性，能够方便的添加新的监控项，来适应网格的迅速发展以及新加入网格的各类资源。Ganglia支持添加自定义的metric，用来收集Ganglia自身不支持的监控数据项。向ganglia加入自定义metric有两种方法：

1) 通过命令行的方式运行gmetric
2) Gmond的metric收集代理提供扩展接口，支持c和python的扩展模块，可以加入自定义的模块来收集特定的监控数据。

用c或者python编写扩展模块的步骤为：

1) 创建一个扩展模块的结构定义，包含回调数据和metric的信息
2) 实现3个回调函数，负责连接Gmond的metric收集代理和metric模块。

这些回调函数包括了模块初始化，metric 处理程序，模块的清理功能。

在此我们通过对进程级监控软件进行二次开发，结合 Ganglia 的监控扩展功能，实现了 Ganglia 不支持的进程级的监控。

1) 要获取进程级的监控状态数据。我们使用 iotop 和 nethogs 两个监控软件，这两个软件分别监控进程对磁盘的读写量和对带宽的占用，只有几十 KB 大小，运行时占用的系统资源非常少，符合网格监控系统的设计要求。

2) 要从规范化的数据中抓取监控系统所需要的监控项。我们使用 sed 和 awk 工具，它们都是一种样式扫描与处理工具，基本功能是对文件进行指定规则浏览和提取信息。

3) 获取到的监控数据加入资源采集模块。我们使用 gmetric 来将获取到的监控信息添加为 ganglia 的监控项。gmetric 是将信息插入到 Ganglia 中的命令行工具。它的主要选项是 name,value 和 type,比如下面的例子，就是在 ganglia 中插入名叫 net_pid 的监控项，监控项的值等于变量 user 的值，类型是 uint32。

6.1.2.4 服务状态采集模块

网格系统中很多功能都是以网格服务的形式来提供的，服务状态采集模块周期性的收集服务状态信息，测试网格服务的可用性，稳定性和正确性，并向聚合层提交测试报告。

- 服务状态采集模块的体系结构

我们采用 TeraGrid 中的 Inca[20]工具来实现服务监控。服务状态采集模块的体系结构图如图 6.2.

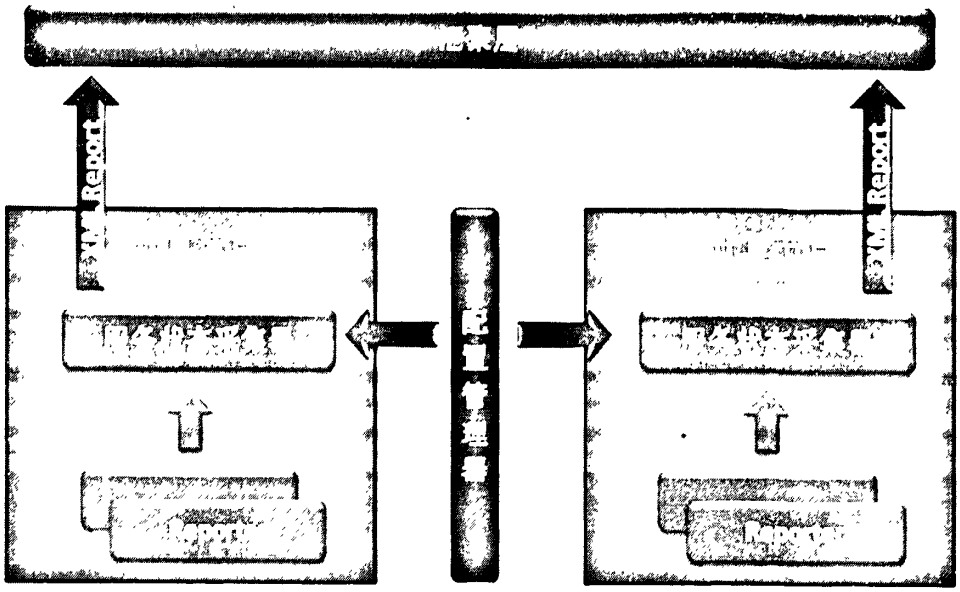

图 6.2 服务状态采集模块体系架构图

服务状态采集器负责周期性的执行节点上的 Reporter。Reporter 是一些可执行的程序，用于测试和测量系统和服务的某些方面的运行状态。配置管理器主要是对 Reporter 和服务状态采集器进行统一的配置和管理。服务状态采集将 Reporter 的执行结构生成 XML 文件，提交给汇聚层。suite 是一系列和某个主题相关的 report 的集合，如数据管理、作业管理、文件传输等等。

- 服务状态采集模块有如下特点：

1) 模拟一个网格用户，运行预先定义的可执行测试脚本

2) 有大量预先定义好的 report 可供使用

3) 测试结果形成 XML 格式

4) 支持对 package 依赖关系的检测

5) 提供集中的配置和管理功能

- 编写一个 reporter

report 通过执行一个单元测试，或者发布一个软件包的最低版本号等方式，来进行对服务可用性正确性的测试。report 可以用任何语言，下面是一个简单的例子，用于监测网络带宽使用的上限和下限。

```
<metric>
```

```
    <ID>bandwidth</ID>
    <statistic>
        <ID>upperBound</ID>
        <value>998.67</value>
        <units>Mbps</units>
    </statistic>
    <statistic>
        <ID>lowerBound</ID>
        <value>984.99</value>
        <units>Mbps</units>
    </statistic>
</metric>
```

6.2 聚合层的设计实现

聚合层为网格各类资源提供统一注册和发布接口，对分布式异构的网格资源、服务、数据的监控数据进行统一的聚合和持久化存储，提供查询和订阅等方式获取聚合的监控数据。

6.2.1 资源注册发布模块的设计与实现

资源注册发布模块为网格各类资源提供统一的注册发布接口，收集和管理注册资源的元数据信息，提供全面的查询接口，为全局资源目录已经资源发现，作业和资源的调度与调配提供数据支持。

6.2.1.1 资源注册发布模块的结构设计

资源注册与发布模块的结构图如图 6.3 所示。分为元数据信息提供源，元数据注册中心和元数据访问接口三大部分。

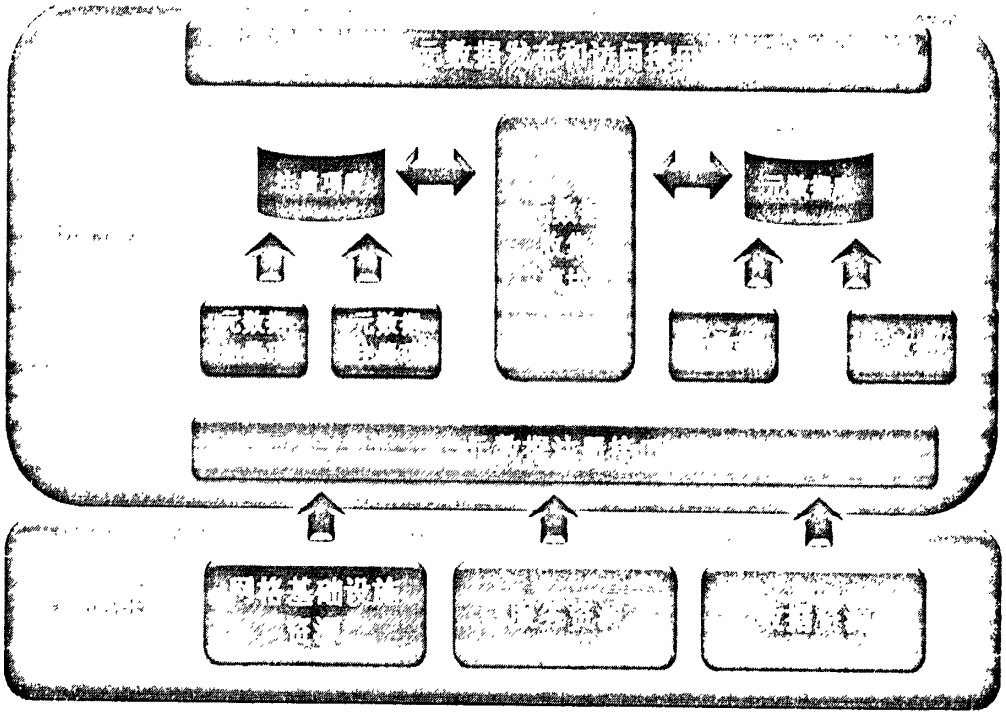

图 6.3 资源注册发布模块结构图

元数据信息提供源是向资源注册发布模块提交元数据信息的数据源。当元数据信息提供源通过元数据注册接口向资源注册发布模块注册之后，资源注册发布模块会根据注册时提供的配置文件，定期更新元数据信息提供源的元数据信息。元数据信息提供源分为网格基础设施信息提供源、服务资源信息提供源、数据资源信息提供源。

元数据注册中心负责对外提供元数据注册接口，已经元数据发布和访问接口，并对元数据注册项以及元数据进行管理和存储。元数据注册中心对每一个注册的元数据信息提供源，都创建一个元数据注册项，注册项中包含元数据信息提供源的注册信息。注册信息通常包括：注册文件的更新周期，可执行程序的执行周期，信息提供源的访问接口，查询资源数据的查询周期，查询资源属性名称等。这些注册项会存入元数据注册中心的注册项库中。元数据注册中心会轮询注册项库中所有注册项，定期会从元数据信息源获得更新的元数据信息。对于获取到的元数据信息，注册中心采用统一的元数据描述模型，对各类元数据进行提取，存入相应的元数据库，并提供元数据的发布和访问接口。

6.2.1.2 资源注册发布模块的工作流程

资源注册发布模块的工作流程如图6.4所示：

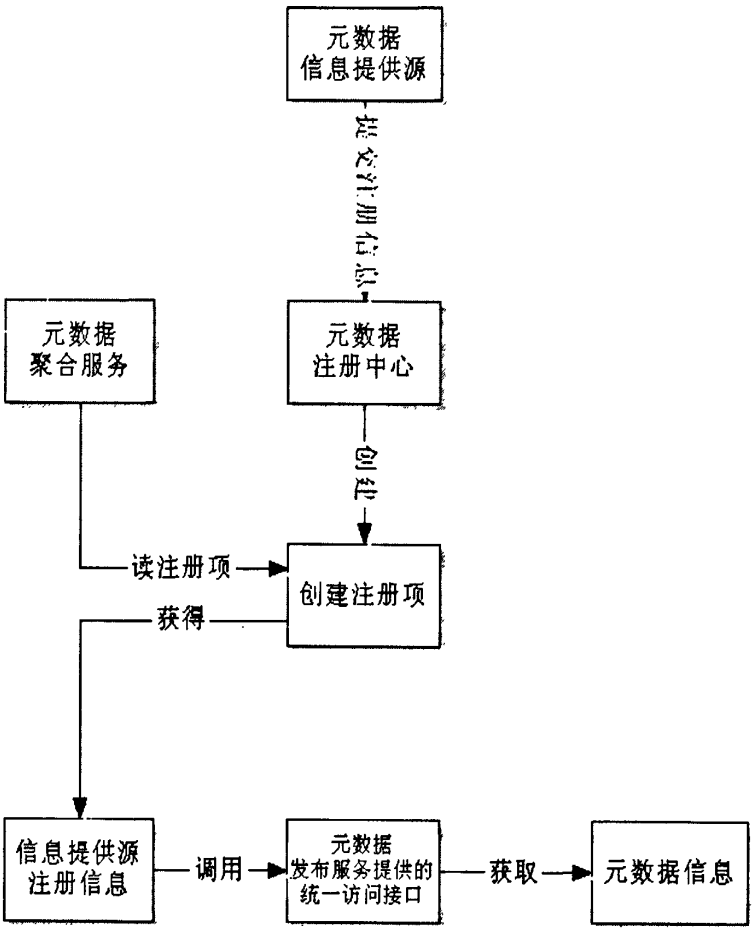

图 6.4 资源注册发布模块工作流程图

1) 配置注册文件

可执行信息提供源的注册文件内容包括：注册文件更新周期，执行程序的周期（执行元数据信息提供源的脚本或程序，获取元数据的周期）、信息提供源的名字（映射到脚本或程序的路径），信息提供源的访问接口等。

查询信息提供源的注册文件内容包括：配置元数据注册web服务的更新周期，查询资源数据的查询周期，查询资源属性名称，信息提供源的访问接口等。

2) 元数据注册

元数据信息提供源向元数据注册系统注册，提交注册文件，从元数据注册系

统获得自己的元数据注册项，注册项包括元数据信息提供源的配置信息，括元数据信息提供源的访问信息等。

3) 元数据聚合

元数据聚合服务读取元数据注册项，获得元数据信息提供源的注册信息，调用元数据发布系统提供的统一发布接口，获取元数据。

4) 元数据访问

提供统一的元数据访问接口，全局资源目录服务调用接口获得资源目录，资源调度服务调用接口，获得合适的资源信息。

6.2.2 资源聚合模块的设计与实现

资源聚合模块主要负责聚合来自资源采集模块的资源状态监控数据，来自任务管理模块的任务进度状态信息，以及其他各个向聚合层提交的数据。

资源聚合模块实现拉和推送(pull/push)两种聚合方式。拉的方式是通过资源注册中心的注册数据，定期向各个资源进行轮询，获取资源的最新状态数据。而推送的方式是当资源的状态属性发送变化时，主动向资源聚合模块推送数据，并且用户可以通过推送方式订阅自己感兴趣的资源数据，当资源状态属性发送变化时，将资源状态信息推送到用户手中。我们主要介绍推送聚合方式的实现。

推送聚合的实现是通过一个软件聚合框架，其最重要的接口分别是AggregatorSource 和 AggregatorSink 接口，符合观察者模式，使得观察者可以高效的获得被观察者产生的数据和更新的数据。

6.2.2.1 AggregatorSource 接口

信息提供源要实现 AggregatorSource 接口，来向聚合模块提供数据。AggregatorSource 接口处理观察者和被观察者之间的会话管理，该接口定义了setAggregatorSink(AggregatorSink sink, java.lang.Object sinkParameters)方法，定义了信息提供源产生的数据将被送给谁，当有新的数据产生，或者数据出现变化时，调用 AggregatorSink 的 deliver (AnyContentType，

AggregatorServiceGroupEntryResource)方法，将数据传递给 AggregatorSink。通过对数据产生和消费的接口的不同实现方式，扩展了信息提供源的多种模式，例如订阅方式，是一种异步的通知模式；查询模式，采用同步的拉数据的方式；还有诸如执行命令行等方式。

6.2.2.2 AggregatorSink 接口

AggregatorSink 接口是聚合数据流出的地方。当信息提供源有新的数据产生时，该 AggregatorSource 会调用注册的 AggregatorSink 的 deliver (AnyContentType，AggregatorServiceGroupEntryResource)方法，将数据传递给 AggregatorSink 接口，因此资源状态数据存储模块，以及其他需要订阅相关信息提供源的模块可以通过实现 AggregatorSink 接口，来收集信息提供源提供的数据。

6.3 展示层的设计与实现

展示层通过一些富客户端技术，以图表，地图等多种丰富的展现形式，对网格系统，集群，节点的运行状态，任务的完成状态等进行可视化展示。使得用户可以更加清晰，直观的从中获取有用的信息。

6.3.1 展示层的功能模块

展示层按照功能不同，分为全局资源目录，资源监控，任务监控，计费管理，统计预测等模块。

- 全局资源目录：当用户登录后，全局资源目录模块提供用户有访问权限的所有资源目录，全局资源目录以树形列表的方式，按照网格、集群、机器节点等多种层次展示资源。通过对资源目录的操作，可以查看到对应资源的具体信息。
- 资源监控：资源监控模块是最重要的部分。主要负责对计算资源，存储资源，网络资源等各种资源的可视化展示，采用图表等丰富直观的展现方式，

可以很方便的看出各个集群节点的运行状态。当有资源出现CPU、内存占用率过高等需要告警的情况时，会用红色将对应的资源标出，并弹出告警对话框。

- 任务监控：任务监控模块可以查看所有提交的任务，正在运行的任务，被暂停的任务信息等。对每个任务，可以查看任务的详细信息，任务完成的百分比等具体信息。
- 计费管理：普通用户可以在计费管理模块中查看自己使用了哪些资源和数据，以及产生的费用。管理员用户可以查看全局资源的使用情况。
- 统计预测：统计预测模块通过对资源监控数据的统计和分析，给出资源和数据使用情况的报表，集群运行状态及稳定性的分析，以及集群后续运行状态的预测。

6.3.2 可视化技术的选用

现在大多数应用大多采用B/S架构，因为B/S架构相对于C/S架构省去了客户端的下载安装配置更新等复杂的操作，因此我们这个监控系统也采用B/S架构。

随着网络带宽基础设置建设的增强，以及富客户端的技术的发展，使得web应用有了更加强大的交互性。Adobe Flex技术是一个高效、免费、富有表现力的开源框架，可以用于构建Web应用程序。Flex通过将MXML和ActionScript编译生成相应的swf文件，具有flash可以展现的丰富的动画效果和交互体验。因此我们采用Flex作为客户端的主要工具。

Flex有丰富的功能强大且质量高的组件和库，在本项目中，主要使用了Birdeye的RaVis库，FusionMap和FusionChart库等多个优秀的组件库。

Birdeye是一个比较庞大的项目涉及数据可视化的关系分析、空间信息分析、数值分析等多个方面。为了满足项目关于节点管理多种视图展示的功能需求，我们使用了Birdeye中的关系分析图表——RaVis。

FusionMap和FusionChart是InfoSoft Global公司的产品，是一套跨平台，跨浏览器的地图组件以及flash图表组件。我们用FusionMap将所要监控的资源

标注在地图上，同时提供基于地理位置的统计分析。用 FusionChart 来做监控数据的展示，由 FusionChart 创建的图表非常美观，流畅。把数据表格转化为图表，更加清晰，直观，易于提取信息。

我们还选用了 Cairngorm 框架来更好的组织 web 端的体系架构。Cairngorm 是一个开源的 Flex MVC 框架。由业务逻辑部分(Business)、命令部分(Command)、控制部分(Control)、数据模型(Model)、界面视图(View)、数据实体(ValueObject)等部分组成。客户端界面是由 View（视图）实现，本项目中将各监控视图都写成一个独立的 MXML 文件，以利于代码复用。

6.3.3 展示层的实现

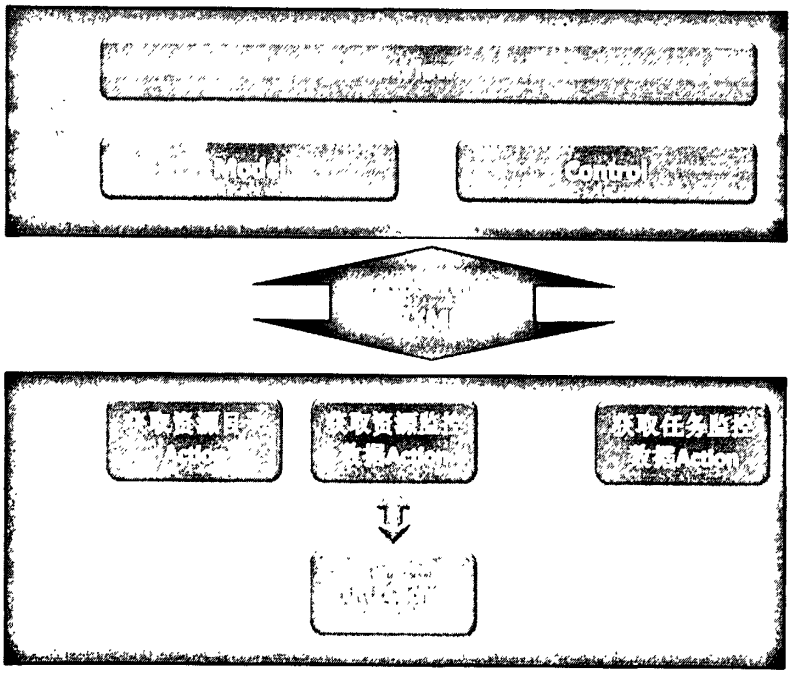

图 6.5 展示层结构图

我们采用 Flex 技术实现 web 端，使用 struts2 框架实现 server 端。

- 全局资源视图的实现

web 端通过调用 server 端的获取资源目录的 action，server 端根据用户信息以及其权限配置，调用服务层获得该用户可以查看的资源目录，并返回给 web 端进行展示。

- 资源监控的实现

资源监控的可视化主要是通过绘制 linechart，piechart 等一些图表，来展示一段时间内（默认是最近两小时）某个监控项的数据变化曲线。资源监控 web 端的流程图如下：

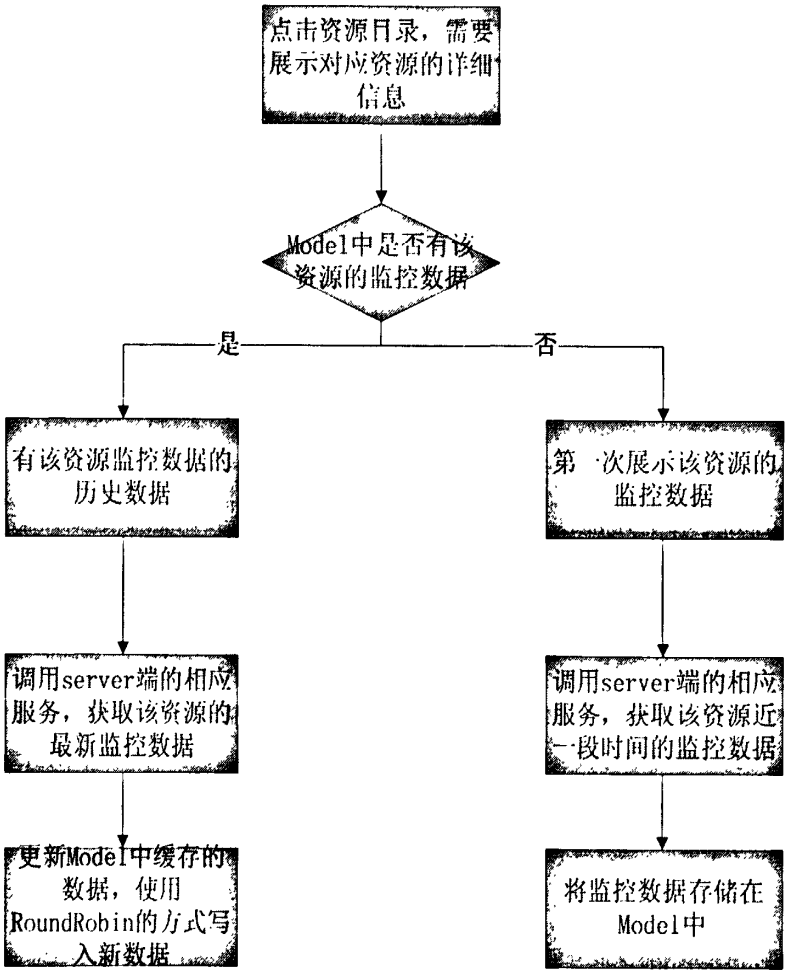

图 6.6 资源监控数据展示的流程图

当 web 端的 Model 中没有该监控项的数据时，及第一次取该数据，web 端调用 server 端的获取资源监控数据的 action，server 端根据 GirdID，ClusterID，HostID，监控项等信息，调用服务层对应接口，获得该监控项这一段时间的历史数据，并返回给 web 端。由于网格系统面向大量用户，所以我们对取到的历史数据进行缓存，这样其他用户再次访问这些数据时，就不需要访问磁盘存储来获取

了。当web端的Model中有该监控项时，说明已有该监控项的历史数据，则只需要向 server 端取最近的数据就可以了。这时要根据历史数据中，时间最近的数据的时间t0，向server端取该监控项t> t0的所有数据。

- 监控数据Cache的实现

对于server端，Cache的使用可以减少对磁盘存储的访问，减少系统io的开销，提高系统的响应速度。尤其是访问用户量大的网格服务，是非常必要的。

对于web端，Cache的使用可以减少对server端的访问，减少网络传输，提高用户的交互体验。

对Cache的设计要考虑两点：一是如何提高命中率；二是控制内存使用。

由于Web端对监控数据的展示是以机器问单位的，所以cache对数据的组织也以机器为单位。通过hashtable来确定cache中某个HostID是否命中，以及该HostID对应数据的index。

由于要控制Cache使用的内存的大小，所以我们采用LRU策略来决定如何替换Cache中的数据。另外，考虑到查看一个cluster整体情况的概率比查看一台机器的概率大得多，所以做LRU策略时，对于cluster的数据和host的数据是分开考虑的。

6.3.4 展示层的可视化效果

图6.7为资源监控的主要操作界面，左边是资源目录，是一个树状的目录结构，可以清晰的看出资源的从属关系。通过点击资源目录中具体的项，右边会展示对于集群或者节点机器的监控数据详细信息。

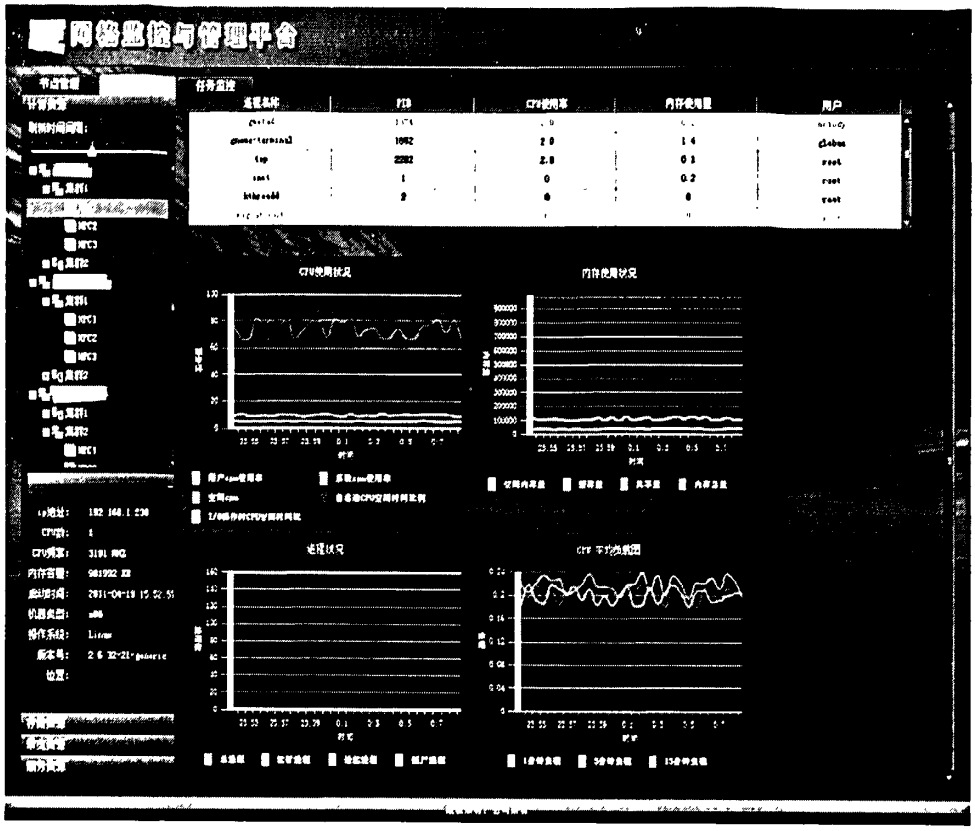

图 6.7 资源监控的主要操作界面

左下角为资源的静态属性面板。展示 IP 地址，CPU 数，CPU 频率，内存容量，启动时间，机器类型，操作系统及其版本号等静态属性信息。

图 6.8 展示了某台机器的 CPU 使用状况，横坐标是时间，纵坐标是百分比，图形按照配置的时间定期刷新，默认是 30 秒钟一次，显示最近 2 小时内的监控数据。包括用户 CPU 使用率，系统 CPU 使用率，空闲 CPU 百分比，自启动 CPU 空闲时间比例，I/O 操作时 CPU 空闲实现比例等多个数据。

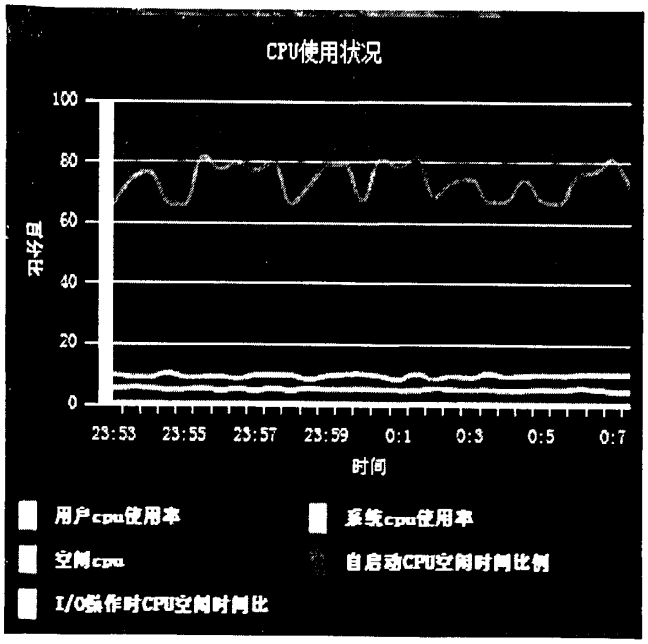

图 6.8 CPU 使用状况监控图

图 6.9 展示了进程级的 CPU 活动情况，使得管理员和用户可以从进程级的角度，了解集群中机器的运行情况。

CPU活动进程：

进程名称	PID	CPU使用率	内存使用量	用户
gmetad	1374	2.0	0.2	nobody
gnome-terminal	1852	2.0	1.4	globus
top	2282	2.0	0.1	root
init	1	0	0.2	root
kthreadd	2	0	0	root
migration/0	3	0	0	root

图 6.9 进程级计算资源监控界面

图 6.10 展示了存储资源的主要监控界面，界面结构和计算资源的监控界面类似，左边是资源目录，左下角是静态属性信息，右面是监控展示主界面。上方的表格是存储资源的进程级使用情况，下面是动态监控图，展示了磁盘 io 随时间变化的曲线。最右边可以收起的面板是一个展示磁盘使用量的 widget。

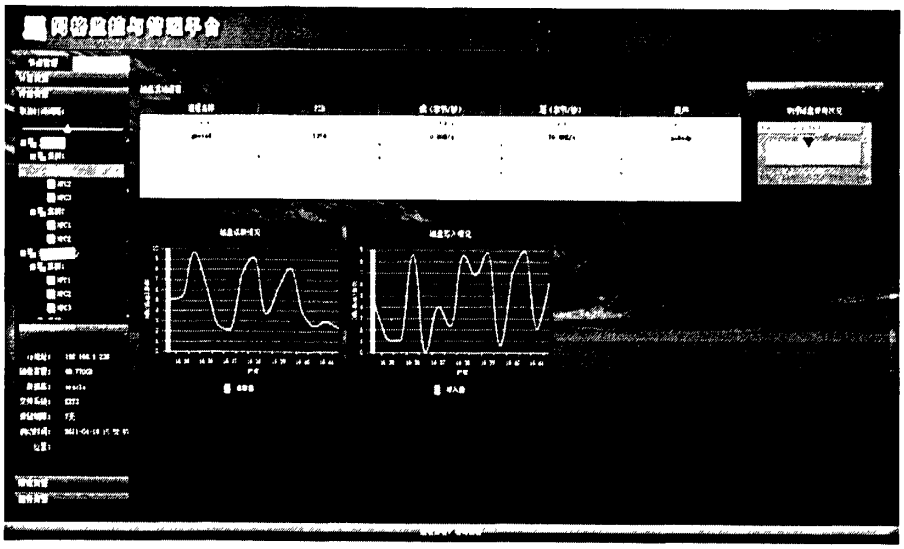

图 6.10 某机器存储资源监控界面

图 6.11 是统计分析的主要界面。上面是功能导航栏，有运行简报，用户统计，作业统计，单位统计，地域统计，行业统计等多种统计功能。

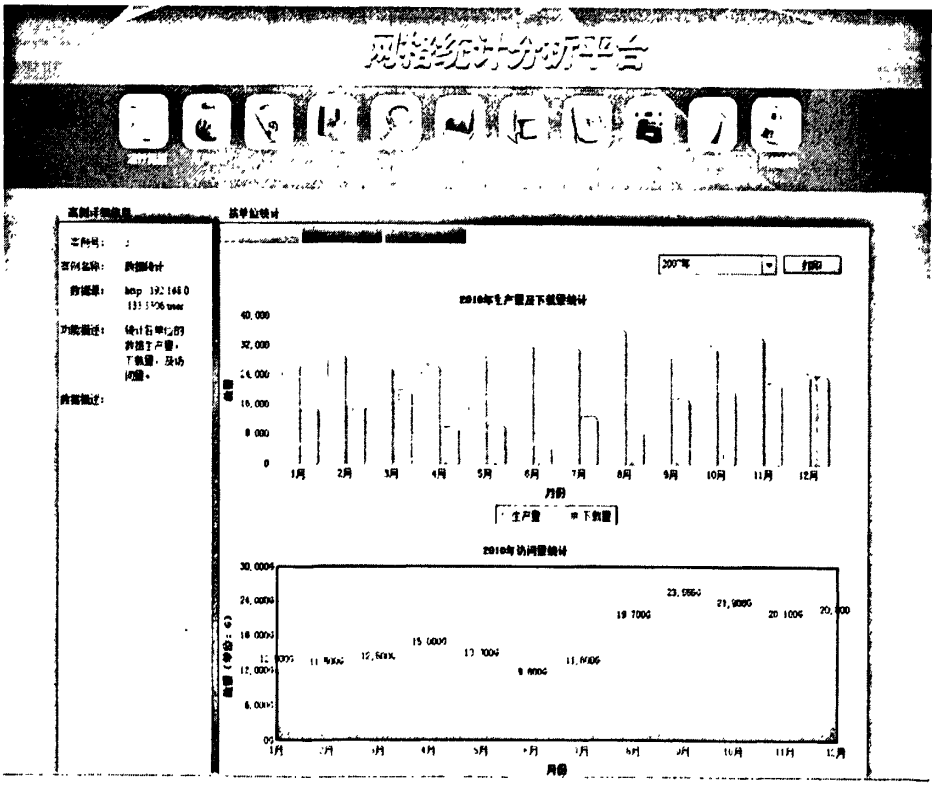

图 6.11 统计分析的主要界面

6.4 本章小结

本章介绍了基于云架构的网格监控系统的详细设计与实现。首先介绍了资源层的监控对象与事件，以及资源采集模块的详细设计与实现。接着介绍了聚合层中资源注册发布模块的设计与实现，以及聚合模块是资源的订阅的实现。最后对展示层的实现进行了详细介绍，介绍了展示层使用的主要技术，详细的描述了展示层实现的难点，并给出了系统的可视化效果图。

第7章 总结与展望

7.1 工作总结

本文对网格监控系统的实现难点进行了分析，介绍了网格监控系统中的三大关键技术，并对基于云架构的高性能大规模网格监控系统的总体设计，以及各个模块的详细设计与实现进行了系统的介绍。

本文的主要贡献有：

1) 本文通过对网格监控系统的实现难点进行分析，提出了基于云计算技术来解决网格监控系统对于海量数据存储，高可扩展性，高可用性和可靠性的需求，实现了一个基于云架构的高性能大规模网格监控系统。
2) 本文提出了基于云架构的存储系统来存储海量的监控数据。解决了以往监控系统只能保持最近一段时间的监控数据，并且监控数据孤立，不能产生更多价值的问题。并且该存储架构能够支持高并发的读写请求，能够提供高效的存储和访问，能够提供丰富的查询和检索方案。
3) 本文提出了基于一致性哈希的高效易扩展资源聚合技术，解决了以往监控系统中的资源聚合框架采用树形架构带来实时性低，容易出现单点故障，不容易进行扩展的问题。该聚合技术能够自动的分配资源与索引服务器的对应关系，易于实现负载均衡，增加或者减少索引服务器对已有结构改变小，同时能够自动检测和处理故障机器。
4) 本文通过使用一些富客户端技术，以图表，地图等多种丰富的展现形式，对网格系统，集群，节点的运行状态，任务的完成状态等进行可视化展示。使得用户可以更加清晰，直观的从中获取有用的信息。解决了以往大多数监控系统使用 RRDTool 绘制图形，对监控数据的展示方式不够灵活等缺点。

7.2 下一步工作

随着互联网技术的发展，网格系统以及云计算系统的集群规模越来越大，对这些系统进行监控和管理是非常重要的。要实现监控系统的高效性、实现性、良好的可扩展性，更好的为用户服务，还有很多工作要做。结合本文的工作，本课题还有很多工作需要进行：

1) 在本文中，我们采用了 MongoDB 作为存储监控状态数据的主要载体，而监控数据有自己的特点，即基本都是追加操作，而没有更新操作。而 MongoDB 中为了适应对文档的更新，为每个文档都预留了一些空间，以防止对文档做小的更新时，需要将文档拷贝到其他放得下的位置。所以这些预留空间在不会出现更新操作的监控系统中，就成为了浪费的设计。下一步，我们希望能够针对监控数据的特点，对 MongoDB 进行优化。

2) 在本文中，我们采用了网格系统 Globus 中的 MDS 作为聚合层的主要框架，下一步，我们希望能够开发一个更加通用的聚合框架，能够适应所有集群系统。

3) 随着移动互联网技术的发展，人们对随时随地能够掌握系统的运行状况提出了需求。现有的监控系统主要是通过向运维人员发送短信息，来发送告警通知。运维人员需要马上赶到公司，或者找到一个可以上网的地方。下一步，我们系统能够在智能终端上提供监控系统的操作终端，这样运维人员可以在手机以及平板电脑上对系统进行一些操作，而管理人员也可以随时随地的查看系统的运行状态，并查看一些统计报表。

参考文献

[1] 吴朝晖, 陈华钧著. 语义网格: 模型方法与应用. 浙江大学出版社, 2008

[2] Francine Berman. 网格计算: 支持全球化资源共享与协作的关键技术. 华中科技大学出版社, 2005

[3] Ruth Aydt, Dan Gunter, Warren Smith, Martin Swany, Valerie Taylor, Brian Tierney, Rich Wolski. A Grid Monitoring Architecture. GWD-Perf-16-1, January, 2002

[4] I.Foster, C.Kesselman. Globus: A Metacomputing Infrastructure Toolkit. Supercomputer Applications. 1997:115~128

[5] Globus mds. http://www.globus.org/mds/

[6] Jennifer M.Schopf, Laura Pearlman, Neill Miller, Carl Kesselman, Ian Foster, Mike D'Arcy, Ann Chervenak. Monitoring the grid with the Globus Toolkit MDS4. Institute of Physics Publishing, 2006:521~525

[7] A.W.Cooke, A.J.G.Gray and W.Nutt et al. The Relation Gird Monitoring Architecture: Mediating Information about the Grid. Kluwer Academic Publishers, 2004

[8] Andy Cooke, Alasdair J.G.Gray et al. R-GMA: An Information Integration System for Grid Monitoring. Lecture Notes in Computer Science, 2003:462~481

[9] DMF. http://www-didc.lbl.gov/DMF/

[10] Sergio Andreozzi, Natascia De Bortoli et al. GridICE: a monitoring service for Grid systems. Future Generation Computer Systems, 2005:559~571

[11] F.Bonnassieux, R.Harakaly and P.Primet, MapCenter: an open grid status visualization tool. Proceedings of the 15th ISCA International Conference on Parallel and Distributed Computing Systems, September 2002

[12] 田鸣华. 网格环境下资源监控系统的研究与实验. 硕士学位论文, 国防科学技术大学, 2004

[13] Baker M, Smith G. GridRM: A Resource Monitoring Architecture for the Grid.

Computer Science, 2002, Volume 2536/2002, 268~273

[14] Federico Sacerdoti, Mason Katz, Matt Massie, David E.Culler. Wide Area Cluster Monitoring with Ganglia. IEEE Cluster 2003 Conference, Hong Kong, December 2003

[15] RRDTool. http://oss.oetiker.ch/rrdtool/

[16] MongoDB. http://www.mongodb.org/

[17] D.Darger, E.Lehman, T.Leighton, M.Levine, D.Lewin and R.Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots On the World Wide Web. ACM Symposium on Theory of Computing, 1997:654~663

[18] Ion Stoica, Robert Morris, David Karger, M.Frans Kaashoek, Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. ACM, 2001

[19] Chord. http://en.wikipedia.org/wiki/Chord_(peer-to-peer)

[20] Shava Smallen, Catherine Olschanowsky et al. The Inca Test Harness and Reporting Framework. IEEE, 2004

攻读硕士学位期间主要的研究成果

[1] 2009.09 - 2010.06

研究课题：中医药数据一体化共建平台

简介：完成了一个高度可配置的通用数据加工系统，实现中医药数据的信息化，为数据挖掘和知识发现提供数据基础。

本人工作：开发人员，参与前后台项目的开发

[2] 2010.04 - 2010.10

研究课题：西苑医院药理数据录入平台

简介：以共建平台为基础，针对西苑医院药理数据录入需求开发了新的平台。尤其是增加了实验数据表格的动态创建，可以灵活添加各种形式的表格。

本人工作：负责人，负责需求调研，系统设计，项目开发与测试

[3] 2010.11-2011.06

研究课题：网格监控与管理系统

简介：该项目实现了大规模网格集群的计算资源、存储资源、网络资源、服务资源的监控和管理，提供了可视化的全局资源监控管理操作界面，为资源发现、作业与资源的调度和分配、统计预测等服务提供数据依据

本人工作：负责人，负责需求调研、总体和详细设计，开发与测试

[4] 2010.09-2010.12

研究课题：一种互联数据查询方法

简介：发明专利。通过定义一套导航约束规则，和一个导航代理装置，对互联数据进行查询，获得查询结果。

本人工作：撰写人

致谢

在论文完成之际，我要向实验室的陈华钧老师和姜晓红老师表示衷心的感谢，他们严谨的治学态度和实事求是的科研精神，对我的学习和科研都有非常重要的影响。

感谢实验室的各位师兄师姐，宓金华、卢宾、王俊健、郑清照、刘洋、罗兆波、黄朝晖、付志红、盛浩、冯叶磊、刘森、杨克特、曹凌、于彤，感谢你们把自己经验总结与传授给我。另外感谢课题组里一直带我的师兄陶金火，感谢你给予我的帮助和鼓励。

感谢课题组的师弟李灼灵，感谢你对我工作的支持和配合。感谢实验室同一级的同学们梁欣颖、王超、郑耀文、刘明魁、张露、彭志鹏、张良杰，感谢你们对我的关心与帮助，大家在实验室营造了良好的学习和工作的氛围，和大家一起感到非常的开心和温暖。感谢实验室的师弟方聪、陈亮、刘东、陈矫彦，在项目组搬到西溪的阶段里，感谢你们给了我很多帮助，也让学习工作以及生活都变得更加有趣。

我将最深切的谢意献给我的父母和家人，感谢他们给了我精神上和物质上的鼓励和支持。

最后，向所有关心和帮助过我的老师、同学和朋友表示最诚挚的谢意

署名 张湘豫

当前日期 2011 年 12 月 19 日

基于云架构的高性能大规模网络监控系统的设计与实现

作者：张湘豫
学位授予单位：浙江大学