清 华 大 学

综 合 论 文 训 练

题目：无极计算中间件中的通信与安全技术研究

系　　别：自动化系

专　　业：自动化

姓　　名：陈硕

指导教师：曹军威　研究员

2012 年 6 月 4 日

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：学校有权保留学位论文的复印件，允许该论文被查阅和借阅；学校可以公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存该论文。

(涉密的学位论文在解密后应遵守此规定)

签　名：_____导师签名：_____日　期：_____

中文摘要

近半个世纪以来，互联网技术与分布式计算技术高速发展。整个 IT 产业逐步转向以服务为中心，以用户需求为驱动的产业方向。在互联网领域，技术的终极目标之一就是让人们无论在任何时间，任何地点，任何设备环境下可以无缝地使用计算资源与服务应用。20 世纪 90 年代，网格计算的出现，使高性能计算与并行计算技术向前推进了一大步。网格计算（Grid Computing）技术将分散在不同位置的计算资源通过虚拟组织的方式组织起来，实现多种资源，包括计算、存储、数据等资源共享。其后，各种计算技术火热发展，其中包括效用计算(Utility Computing)，集群计算(Cluster Computing)，云计算(Cloud Computing)等。

在这个趋势下，网格技术的逐渐发展成为所谓的“cyberinfrastructure”,即赛百平台（CyberInfrastructure，CI）。人们已经不再像以前一样希望创造一个集硬件、软件、网络和服务的计算基础设施。无极计算(ELOP Computing)是由清华大学信息科学国家实验室曹军威研究员对未来计算机发展提出的一种开创性和前瞻性的构想，旨在致力于解决赛百平台和网格计算技术在纵向深化发展的过程中出现的一些诸如元素大量分布，资源动态异构，应用位置独立等遇到的实际问题和上节中提及的诸多应用需求无法满足的困窘。基于此目标提出了 ELOP (Element, Logic, Organization, Process)的四层计算架构。

本文首先介绍了无极计算 ELOP 架构与基于此架构开发的 ELOP 异构资源管理中间件的总体结构。然后具体介绍了 ELOP 软件包中的通信系统、安全系统与无极计算中间件的系统实现的技术框架。在文章的后半部分具体介绍了对于以上问题的具体解决方案，技术实现与运行性能。

关键词：无极计算、分布式计算、互联网、SSL、X.509

ABSTRACT

Over last half century, internet and distributed computing technology greatly developed. The whole IT industry has turned into service oriented industry and driven by the needs of customer. In the area of internet, one of the ultimate objectives is to access to any service on the internet anywhere and anytime. In 1990s, the grid computing evolved in distributed computing and took effects in the sharing computing resources over the internet. As follows, various grid-like technologies emerged such as Utility Computing, Cluster Computing and Cloud Computing.

In this trend, grid computing has become infrastructure of computing resources. The hardware and software is now distributed over the internet as infrastructure. Thus, the ELOP Computing architecture is proposed by Professor Junwei Cao from Tsinghua University. It is designed to organize and share heterogeneous computing elements over the internet and offer service as infrastructure.

In this paper, we firstly introduce the meaning and architecture of ELOP Computing. Then we will illustrate the development of security system and communication system of ELOP Computing middleware. We will introduce the technology and the process of realization of ELOP Computing middleware. At last, we will show the performance of our ELOP system.

Keywords： ELOP Computing Distributed Computing Internet SSL X.509

目　　录

第1章 序言

1.1 无极计算产生背景

近半个世纪以来，互联网技术与分布式计算技术高速发展。整个 IT 产业逐步转向以服务为中心，以用户需求为驱动的产业方向。[1]在互联网领域，技术的终极目标之一就是让人们无论在任何时间，任何地点，任何设备环境下可以无缝地使用任意规模的计算资源与服务应用。我们可以设想这样的一个场景，一名中国气象科学家在实验室里面执行一个科学计算任务，需要联合若干台超级计算机才能完成这项任务。他预计 3 小时会计算完成，但是他想在家中的 ipad 上查看实验结果，并想将结果分享给美国的同行。对于这样实际的需求也就是所谓的 Anytime，Anywhere，Any Application(A4)问题，在传统的集中式的计算架构当中，A4 问题无法得到很好的解决。

科学界对于高性能计算的需求促进了分布式计算技术的发展。在 20 世纪 90 年代，逐渐产生了网格计算的雏形。一些科研项目尝试连接与整合分散在各地的计算资源以获得更大的计算能力，消除信息孤岛。网格计算的概念首先由 Ian Foster 教授提出[2]，将其定义为“动态的、多机构虚拟组织之间的受控协同资源共享以及问题解决”。网格计算整合了异构的计算资源、包括 CPU 计算能力、存储器、数据、存储等资源，为科学界的复杂的计算提供了重要的技术支撑。一些国外著名的网格平台项目包括 TeraGrid、CoreGRID,EGEE 等，应用领域十分广泛，包括天体物理、地球科学、医学、大气科学、生物信息学等等[12]。网格技术的带动了并行计算技术的火热发展，其中包括效用计算(Utility Computing)[3]，集群计算(Cluster Computing)[4]，云计算 (Cloud Computing)[5]等。

在这个趋势下，网格技术的逐渐发展成为所谓的“cyberinfrastructure”,即赛百平台（CyberInfrastructure，CI）[6]。人们已经不再像以前一样希望创造一个集硬件、软件、网络和服务的计算基础设施，取而代之的是对新的数据重用，以及对所产生数据的长期管理和利用的兴趣。像电网，无线通信网络等其它领域一样，计算资源的基础架构化已经成为一个必然趋势。但是现有的计算模型

与架构在很大程度上并不能很好满足日益增长的应用需求与多变的工作流程。并且在系统的可扩展性、可靠性、安全性、高性能性方面一直没有得到很好的满足。

1.2 无极计算的核心理念与基本结构

1.2.1 无极计算的核心理念

无极计算[13]是 ELOP (Element, Logic, Organization, Process) Computing 的中文翻译，旨在致力于解决赛百平台和网格计算技术在纵向深化发展的过程中出现的一些诸如元素大量分布，资源动态异构，应用位置独立等遇到的实际问题和上节中提及的诸多应用需求无法满足的困窘。网格计算的目的是将计算资源与存储分离开来，并能够动态地调度整合。无极计算则更进一步地，以基础架构的方式来整合异构的计算资源，彻底地将计算主机，数据，软件应用，显示，变量等松耦合的元素彻底地分离开，在 Internet 上进行统一地整合调度。

无极计算的核心思想来自于基础架构思想。基础架构（Infrastructure）[9]一般指支撑一个社会、公司、服务或设施经济运行的基础物理或组织结构。典型的基础架构包括道路交通、供水系统、电力系统、无线通信、互联网等等。各个基础的服务领域在最后都是以基础架构的形式来提供服务。各个领域的基础架构从不同的方面对社会的运转起到了支撑作用。[8]

为了适应 21 世纪的知识经济社会，信息技术的发展以及尤其是分布式计算与通信技术领域的爆炸式增长，使得社会产生对于新的基础架构的需求。美国国家自然科学基金(NSF)由此提出了赛百平台的构想(*cyberinfrastructure* CI)。

随着网格计算技术与云计算技术的发展，资源的基础架构化是一个无法回避的趋势。网格计算更加关注于不同领域资源共享，但是应用与服务却无法很好地进行分布共享。而且网格计算的主要应用还是局限于科学研究领域的高性能、高吞吐量的计算。对于大多数的计算应用需求来说，一般都是大量的、异构的、小规模的计算服务。这些服务应该在任何时间，任何地点，任何的设备条件下都应该很好地得到利用。云计算更加关注与提供服务，但是由于骨干数据中心如果不进行基础架构化[11]的话，则无法进行大规模的扩展。这就需要一个全新的基础架构来提供一个统一的框架，同时将资源基础架构化与提供服务结合起来。因此我们从基础架构化的角度提出无极计算框架。

1.2.2 无极计算架构与内容

无极计算是对赛百平台进行的一种更加详细和具体的解释[7]，是一种面向具体领域的应用计算平台。无极计算最重要的一个原则就是将异构的资源进行标准化与封装，以此能够根据不同需求，动态地整合与加载不同的资源。同时，无极计算中的各个元素都是在互联网上分布式地加载，主机、数据与存储、软件与显示（远程连接或者虚拟桌面）能够解耦与分离。为了能够实现任何时空，任何设备使用任何服务，将资源进行抽象，使其能够在逻辑上进行统一调度，并在更高的层次上进行工作流级别的整合与调度。在资源使用安全与权限方面，增加了数字证书加密通信与虚拟组织的机制。

无极计算的基本架构为四层结构。

1) E (element)元素层：元素层是无极计算基础架构的最底层，其中定义了无极计算中的框架中独立的基本调度及独立应用服务单位元素(element)。对外提供统一规范的通信与调度接口，对内具体封装异构的资源，实现内外的操作对接以及物理描述规范。
2) L(logic)逻辑层：逻辑层实现无极计算任务与应用逻辑。将底层大规模异构的元素(element)屏蔽起来，将元素的物理位置信息与具体的位置进行解耦，在网络层面实现无差异化的逻辑描述，对元素提供统一的访问操作机制。因此用户当应用到某个元素的时候，不用关心元素的具体物理属性，只要按照同一个规范的操作接口即可实现对元素的操作。同时逻辑层面对外提供调度接口，用户可以根据自己的应用需求，在逻辑层面上采用需要的算法或方法，实现对大量异构元素逻辑调度。
3) O(organization)组织层：组织层实现无极计算的访问与操作权限控制，并对无极计算中的元素，逻辑与用户实现统一的管理。通过对 CI 中心实现对于整体 ELOP 框架进行权限与证书的分发与管理。通过类似网格计算中的虚拟组织(virtual organization)来对用户操作元素实现权限管理，在虚拟组织中，用户可以共享元素的操作权限。
4) P (process)流程层：流程层是无极计算的最上层，实现直接和业务逻辑相关的操作与处理。本层只需关心具体的服务或者工作流程而不需要终端用户关心实现细节。在本层中，用户可以实现对于具体应用的流程级聚合，检测元素的运行状态，并向外界提供 ELOP 架构的操作接口。

无极计算架构的简图如图 1.1，图中展示了无极计算的四层架构，并说明了每一层具体功能。

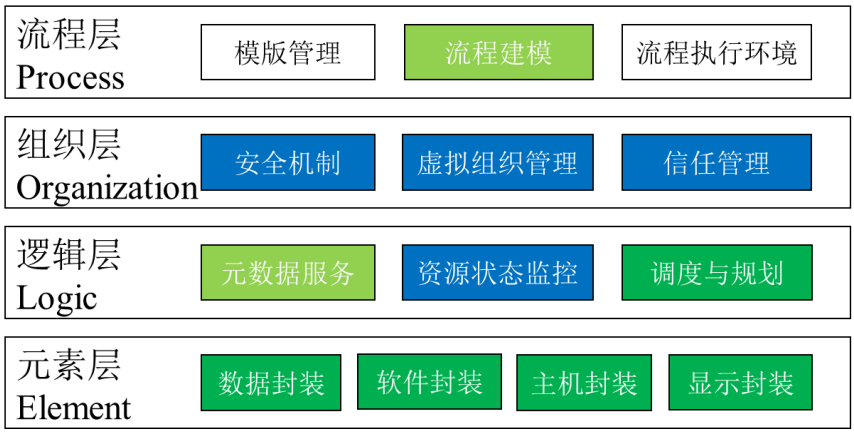

图1.1　无极计算架构示意图

1.3　本文内容及目标

本文主要关注无极计算架构在实现过程中，及无极计算软件包开发过程中所涉及的一系列问题及解决这些问题的技术手段和具体实现，总体包括以下两点。

1）安全问题：无极计算软件包开发过程中所采取的安全解决方案，其中涉及到安全认证、权限管理与加密通信等一整套安全机制。在具体实现中采用了 SSL 安全协议与 X.509 安全证书的加密通信。

2）通信问题：无极计算中间件，采用 socket 通信，支持两种通信方式，同步通信与异步通信。同时可以增加 SSL 加密通信支持。

3）程序管理：无极计算软件包的打包问题，其中涉及到软件包开发过程中的编译与安装管理，用户可以根据自己的需求来安装自己想要的程序版本。将编译与安装过程简单化，用户不需要考虑复杂的环境检测与设置问题，将这些交与程序自动化完成。

1.4　文章结构安排

在本文的第 1 章中，介绍了本文的背景，即无极计算的产生背景与具体架构。同时介绍本文及毕设工作中的主要工作，即开发无极计算软件包开发过程中所遇到的一系列问题与采用的解决方案。

在第 2 章中，我们将对开发过程中所采用两个关键的技术：SSL 安全协议与 X.509 证书认证；异步通信机制；Autotools 软件包管理软件。文中将结合无极计算架构及项目需求的特点来分析所采用技术的优势与劣势。

在第 3 章和第 4 章中，本文将根据第 2 章所叙述采用的技术，对无极计算软件包中的安全通信与同步通信，异步通信分别进行总体设计。随之介绍各个模块在开发过程中的具体实现。同时在各章之后还将介绍在实际开发过程中碰到的具体问题与解决方法。

在第 5 章中，将介绍无极计算程序的具体实现，主要介绍安全通信与同步异步通信的整合，基于安全与通信如何去开发其它层的程序，以及如何打包程序，对整个程序进行编译与安装管理。

第2章 关键技术分析

2.1 安全与权限管理系统

在无极计算软件包开发框架中，在安全方面[17]，采用 SSL(Secure Socket Layer)[18]作为加密协议，对通信实行安全加密。具体加密算法采用 RSA 的非对称加密算法来实现。在 RSA 算法实现过程中，产生公钥私钥对，公钥采用统一的 CA(certification authority)来进行管理。公钥以 X.509 证书的形式对外发布，并通过证书的唯一性对整个无极计算软件包系统的各个单位来进行权限管理。在编程实验之中通常采用 RSA 算法。

2.1.1 非对称加密技术

非对称加密也称作公钥加密(Public-key cryptography)在加密过程中需要两个密钥，公钥和私钥(public key and private key)。在加密通信的应用当中，通过公钥对所需加密的内容进行加密，通过公钥来进行解密。公钥在对外进行发布，私钥则被用户进行保存。在加密通信系统中，公钥用来对文本进行加密，私钥用来对文本进行解密；在数字签名系统中，私钥用来对文本进行加密，公钥用来对文本进行解密。本项目中只关注加密通信系统。

非对称加密通常采用非对称密钥算法，无法通过私钥去推导出公钥的信息，也无法通过公钥去推导出私钥的信息。通常非对称算法通过一个数学难题来保证加密的可靠性，比如大整数的因数分解等。虽然无法互相推导出来，但是公钥与私钥在数学上是相关联的。公钥用来加密信息，私钥则用来解密。因此通过公布公钥，任何人都可以向持有私钥的用户来传递加密过后的信息，而不会被监听。

由于公钥与私钥不可以互相推导出来，因此不像对称加密算法，公钥在交换发布过程中不需要经过安全的通道，虽然有以上诸多优点，但是公开密钥算法的缺点也比较明显。由于通过数学难题来进行加密，所以加密与解密运算的复杂度比较高，运算速度比较慢。同时密钥的制作、发布与管理也是一个需要谨慎解决的问题。

2.1.2 RSA 加密算法

RSA 加密算法是公钥加密算法的一种，它的前提假设是因式分解大整数是一个十分困难的问题。本项目在加密过程中就采用了这种算法。RSA 算法的名字代表了其发明人 Ron Rivest, Adi Shamir 和 Leonard Adleman，他们首次在 1978 年联合发表了该算法。当前大多数服务器客户端的系统也都采用了该算法。

2.1.3 X.509 数字证书

数字证书是网络通信中认证身份的一种方式，同时也是在公钥加密系统中发布公钥的一种规范。传统的证书发布经由 CA(certification authority)发布与管理。在本项目中，我们采用 X.509 格式的数字证书。X.509 证书目前使用非常广泛的证书格式，其符合 ITU-T 的 PKI 与 PMI 标准。在 X.509 系统中，需要 CA 中心来根据公钥来签发证书，将证书与特殊的可识别的名字绑定。

一个典型的 X.509 证书如下所示，其中包含了证书所采用的 X.509 的版本号，加密算法，公钥算法，公钥内容，发布者签名，证书具体内容等。第一版证书的详细说明可以参见 RFC 1422。

```
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 3 (0x3)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=AU, ST=Some-State, O=Internet Widgits Pty Ltd,
OU=elop
        Validity
            Not Before: Aug 24 06:54:10 2011 GMT
            Not After : Jan  9 06:54:10 2039 GMT
        Subject: C=AU, ST=Some-State, O=Internet Widgits Pty Ltd,
CN=E1
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (1024 bit)
                Modulus:
                    00:b6:1a:6e:68:5b:38:b2:bb:d1:1f:1e:0d:08:57:
                    10:8b:0a:78:ae:ba:36:47:83:59:2a:4a:83:a0:83:
                    8d:92:3a:c7:c5:d4:6e:4c:54:c0:8b:44:39:39:f3:
                    62:1b:70:81:5e:97:31:bf:38:5c:8e:0c:15:24:ca:
                    25:4e:64:80:20:86:25:ef:82:3d:67:ca:1d:6b:1f:
                    54:5e:dd:eb:40:e0:b7:8e:f5:ec:5e:59:5c:73:0b:
                    0e:bc:4f:f3:7d:70:46:5b:c1:dd:78:10:d0:ff:50:
```

81:be:9b:11:4b:8c:ef:86:11:19:47:24:70:3a:38:
35:28:1e:d0:20:33:dd:b7:33
Exponent: 65537 (0x10001)
Signature Algorithm: md5WithRSAEncryption
47:db:0c:bb:75:16:a0:24:c2:5c:53:80:b9:95:b5:55:6e:e7:
6a:07:fc:f6:ea:ca:d5:a6:de:43:6f:b6:4c:12:a5:d8:b7:c5:
43:9e:dc:13:46:f7:6b:ad:bc:a5:a3:d2:55:53:a5:1a:bc:3c:
42:8e:f8:4e:b1:5e:f9:b8:55:b2:5d:04:f6:9c:d5:dc:e6:3d:
ae:c1:7a:07:4c:29:d6:ae:35:4d:d5:7c:fd:e9:62:af:dc:ba:
4b:79:a2:45:1d:65:48:ce:3d:6a:fb:54:47:16:c7:9c:d6:32:
d0:96:59:e4:38:98:d5:64:97:38:1e:ef:28:96:8a:ea:16:ef:
74:a7
-----BEGIN CERTIFICATE-----
MIICFTCCAX4CAQMwDQYJKoZIhvcNAQEEBQAwVDELMAkGA1UEB
hMCQVUxEzARBgNV
BAgMClNvbWUtU3RhdGUxITAfBgNVBAoMGEludGVybmV0IFdpZGdpd
HMgUHR5IEx0
ZDENMAsGA1UECwwEZWxvcDAeFw0xMTA4MjQwNjU0MTBaFw0zOT
AxMDkwNjU0MTBa
MFIxCzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMS
EwHwYDVQQKDBhJ
bnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQxCzAJBgNVBAMMAkUxMIGf
MA0GCSqGSIb3
DQEBAQUAA4GNADCBiQKBgQC2Gm5oWziyu9EfHg0IVxCLCniuujZHg
1kqSoOgg42S
OsfF1G5MVMCLRDk582IbcIFelzG/OFyODBUkyiVOZIAghiXvgj1nyh1rH1
Re3etA
4LeO9exeWVxzCw68T/N9cEZbwd14END/UIG+mxFLjO+GERlHJHA6OD
UoHtAgM923
MwIDAQABMA0GCSqGSIb3DQEBBAUAA4GBAEfbDLt1FqAkwlxTgLm
VtVVu52oH/Pbq
ytWm3kNvtkwSpdi3xUOe3BNG92utvKWj0lVTpRq8PEKO+E6xXvm4VbJd
BPac1dzm
Pa7BegdMKdauNU3VfP3pYq/cukt5okUdZUjOPWr7VEcWx5zWMtCWWeQ
4mNVklzge
7yiWiuoW73Sn
-----END CERTIFICATE-----

2.1.4 TLS 与 SSL 安全协议

安全传输层协议(Transport Layer Security)，及其上一代安全套接层(Secure Sockets Layer)是互联网上的一整套提供加密通信的协议。目前几乎所有的 Web 服务器都对 SSL 协议提供支持。SSL 协议的结构如下图所示，SSL 协议处于应用层与传输层之间。其中 SSL 握手协议主要用来处理 SSL 通信开始前的握

手操作。在握手的过程中，通信的双方交换证书已验证身份，之后确定双方之间所采用的加密算法，之后双方交换用于加密传输中用来加密的密钥。握手完成之后，双发即可在传输层利用刚才交换的密钥进行加密传输。

在本项目中，我们还是采用非对称的加密算法 RSA 来进行加密通信。

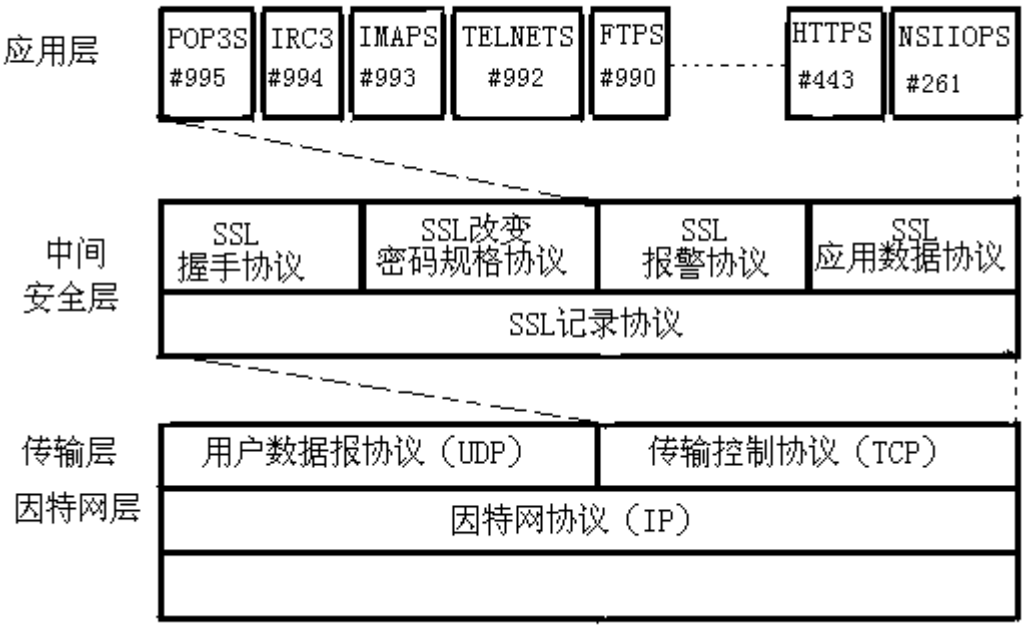

图2.1　SSL协议系统架构

2.1.5 OpenSSL 加密通信开发工具

本项目采用开源的 SSL 与 TLS 开发源代码 OpenSSL 进行开发，该开源代码利用 C 语言进行开发，将 SSL 通信中的基础加密函数进行了封装，提供了动态链接库来调用相关函数。　并且 OpenSSL 可以跨平台地在类 UNIX 系统中应用，大大地减轻了开发 SSL 程序的实现难度。本项目就采用 OpenSSL 中有关 X.509 证书与 RSA 加密算法的函数。

2.2 同步通信与异步通信

同步通信与异步通信的差别主要体现在程序执行的阻塞与非阻塞方式。具体来说，在同步通信的过程中，发送方发出数据后，只有当接收方确认接收成

功以后才可以进行下一次通信操作。异步通信则是发送方发出数据后，不需要等待接收方的接收成功信号，就可以进行下一次通信操作。

在时序方面，同步通信需要保持通信操作的时序同步。也就是说发送端发出 A 通信，接收端则要回应 A’，之后发送端发出 B 通信，接收端则要回应 B’；异步通信不需要保持时序，也就是说发送端发出 A 和 B 通信后，接收端可以先处理 B’再处理 A’。

在通信效率方面，由于同步通信一直要保持 socket 畅通，socket 占用时间过长。而异步通信只要完成通信后，马上就可以关闭 socket，相比之下网络带宽资源占用就要小很多。

在多用户并发方面，同步通信和异步通信都可以做到多用户并发。同步通信中采用的是每进行一个通信连接则多开一个线程的方式。异步通信中只需要维护一个事件消息队列，不断地判断消息队列中是否有新事件，来接收不同发送端的连接请求。相比于多线程同步通信，异步通信无论从资源的节省度和效率来说都有优势。

2.3 Autotools 程序打包工具分析

GNU 编译系统，同时也加 Autotools[19]，是一种帮助在类 Unix 上编译程序源码的一系列编程工具。

在我们进行手动编译 C 源代码，或者自己写 makefile 的时候，有很多情况很难处理：不同系统上不同的编译器；在某些系统上缺少某些链接库。这些问题可以利用条件编译来解决，例如在程序文件中加入很多条件编译块(#ifdef)；但是当编译变量过多时，代码量膨胀遍无法控制。Autotools 的诞生便是为了来解决以上问题的。

Autotools 属于 GNU 工具链中的软件，并在免费软件与开源软件中得到了广泛的使用。

Autotools 包含了 GNU 三大工具软件：Autoconf，Automake 和 Libtool。

2.3.1 Autoconf

Autoconf 根据文件 configure.ac 生成 configure 脚本。在 configure.ac 中描述了源代码的整体属性。生成的 configure 脚本在运行过程中首先扫描系统的环境变量，并生成 config.status 脚本将环境变量与 Makefile.in 文件结合，来生

成 Makfile 文件。此时 Makefile 文件已与系统环境进行了整合。最后通过 make 指令，根据 Makefile 来编译源代码。

GNU 编译系统的复杂程度反映了源代码被编译时所涉及的系统环境变量的数目。

- 如果源代码更改了的话，只要重新运行 make 指令编译部分程序代码即可
- 如果更改了 Makefile.in 文件的话，只要重新运行 config.status 将环境变量重新写进 Makefile.in 生成 Makefile，之后运行 make 指令即可。
- 当源代码拷贝到其它系统的时候则需要重新运行 configure 脚本，扫描环境变量生成 config.status，并重新运行 make 指令。也因此 GNU 软件包发布的时候通常不带有 configure 脚本所生成的文件。
- 如果源代码发生重大更改的话，则需要重新编写 configure.ac 与.in 文件，之后便运行类似的后续步骤。

在执行 Autoconf 的过程中，GNU 系统应用了 m4 宏系统。

随 Autoconf 发布的还有一些辅助工具包括 Autoheader，用来帮助管理 C 程序的头文件；Autoscan 用来创建 Autoconf 的程序模板；ifnames，可以用来列出 C 程序的预编译变量。

2.3.2 Automake

Automake 工具以 Pearl 语言写成，主要包含以下两个指令工具：aclocal 和 automake。Automake 工具并不是向下兼容的。例如用 1.4 版本的 Automake 编译出的程序，用 1.9 版本的则未必能够编译通过。

Aclocal 根据 configure.in 文件生成 aclocal.m4 文件。Aclocal 首先扫描所有的.m4 文件与 configure.in 文件中定义的宏，将这些宏重新定义在 aclocal.m4 文件之中。

Automake 用来帮助编程者用一种更加高级的方式通过编写 Makefile.am 来编写 makefile 文件，而不是通过纯人工来编写 makefile 文件。Makefile.am 需要提供：

- 一行用来标注需要编译出来的程序的名字
- 涉及的源程序文件列表
- 编译器需要的编译选项， 其中包括所需要头文件所在的文件夹路径
- 连接器需要的连接选项，其中包括需要的链接库与其所在文件夹的路径

根据以上提供的信息，Automake 工具生成一系列 makefile 文件，可以让用户来：

- make——编译程序
- make clean——清空编译出来的一系列文件
- make install——安装程序到标准文件夹
- make uninstall——在安装位置卸载已安装程序
- 创建代码分发包
- 测试代码包是否完整，特别地可以在非代码位置里编译程序

Automake 同时会处理软件编译过程中所自动产生的依赖性信息。当程序源代码修改的时候，下次 make 编译的时候，系统会知道到底是对哪些文件进行了更改。如果系统允许，Automake 会动态地更改依赖关系表。Automake 会根据绑定的 decomp 脚本来生成依赖关系。如果编译器是最近版本的 gcc 编译器，automake 会通过内联将依赖信息直接加进 gcc 中，来直接引入。

2.3.3 Libtool

Libtool 是通用库的支持脚本。Libtool 将编译共享库复杂的过程隐藏起来，并能够提供稳定与统一的接口。

过去，如果开发者想开发共享库的话，它需要写大量的支持代码以能够在不同的平台上运行。同时，也需要给库文件的安装使用者提供一个配置文件来让其选择编译与安装库文件的种类。

Libtool 简化了开发者的工作，通过将平台信息与用户接口进行了封装。Libtool 可以在任何的主机与系统上通过一个统一的接口运行，很多复杂的工作责备隐藏了起来。

Libtool 生成的共享库接口是统一的。用户不需要去阅读底层的文档来用共享库去编译别的东西。只需要运行 configure 程序即可，而那些细节问题则都由 Libtool 工具解决了。

2.3.4 利用 Autotools 编译程序的总体过程

利用 Autotools 编译的总体过程如图 2.2 所示。主要步骤为利用 autoscan 程序生成 configure.ac 的程序模板。执行 aclocal 生成 aclocal.m4，运行 autoheader 生成 config.h.in，运行 automake 将 Makefile.am 文件生成 Makefile.in 文件。运行 autoconf 根据 configure.ac 生成 configure 脚本。运行 configure 脚本生成

config.status。运行 config.status 生成 Makefile 文件；之后运行 make 与 make install 即可对程序源码进行编译与安装。

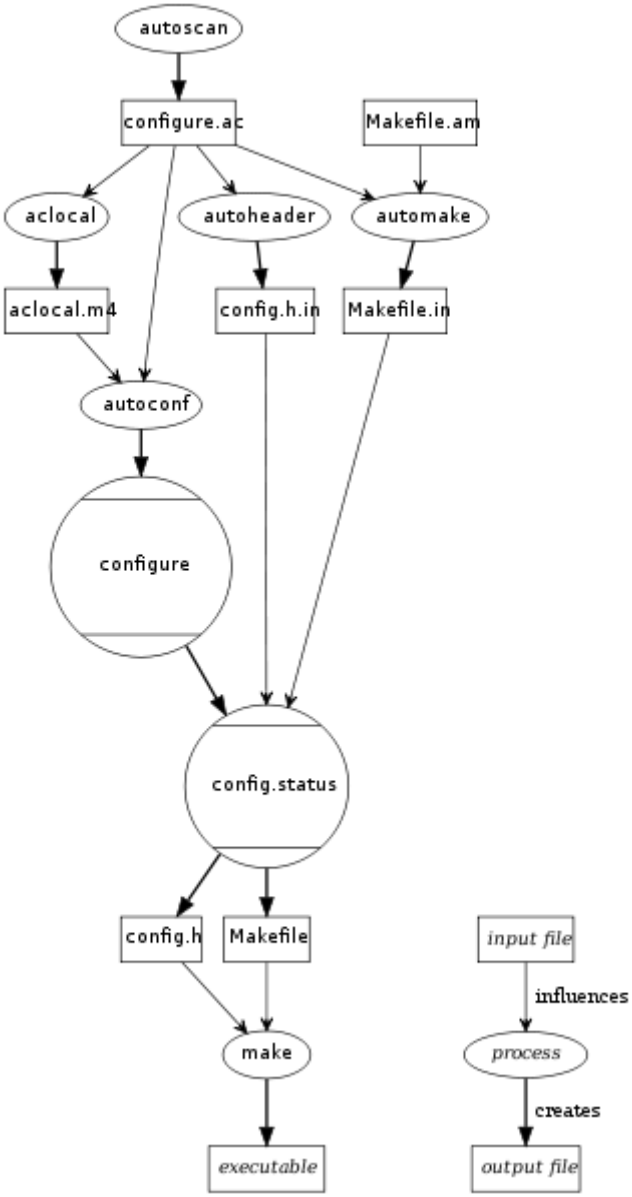

图2.2　Autotools工具编译程序流程图

第3章 无极计算中间件安全认证与通信系统开发

3.1 安全认证与通信系统概述

在无极计算过程中，对于系统安全的需求主要来自两方面，一个保证通信过程的安全性，即加密通信；另一个是对于不同元素与用户之间的身份和权限管理。

在加密通信中，我们采用 SSL 安全协议来保证通信的安全性。为了实现不同元素与用户之间的加密通信，主要需要实现如下问题：

1. 无极计算中各个服务器与用户证书的生成与管理。在本系统中采用 CA 认证中心的方法。首先由 CA 生成公钥和私钥来制作根证书，其它证书的公钥和私钥都由 CA 中心来签发。在 CA 中心建立数据库来保存公钥。
2. SSL 通信的实现：在此过程中需要对证书合法性，有效性和同一性进行验证。合法性即是否为同一 CA 所签发，有效性即验证证书是否过期，同一性及验证证书所代表的身份与所匹配的公钥是否匹配。其中合法性与有效性可以，在 SSL 通信过程本身来实现，和同一性的验证需要与 CA 中心数据库公钥数据库来进行验证。

3.2 安全系统的具体实现

3.2.1 CA 系统实现以及证书生成与管理

下图为 CA 的系统架构[16]设计图，如图 3.1 所示。其中密钥服务器用来生成公钥，私钥。证书服务器用来将生成的公钥制作成证书，并用来储存证书与身份的对应关系。请求处理服务器主要用来处理新的单位要生成证书的请求，以及证书的分发。

当用户或者新的无极计算应用服务器（包括虚拟组织服务器，逻辑服务器与各元素服务器）注册进无极计算系统时，便向请求服务器发起请求申请证书，请求服务器收到单位的相关信息以及证书请求之后，请求密钥服务器根据单位信息生成公钥和私钥。密钥服务器将公钥与单位信息发给证书服务器用来生成证书。本系统中证书服务器用来生成的是 X.509 格式的证书。证书服务器同时

建立数据库来储存证书与用户信息一对一的关系。新生成的证书在请求服务器上通过邮件，以及硬件储存器拷贝的形式来分发证书与私钥。

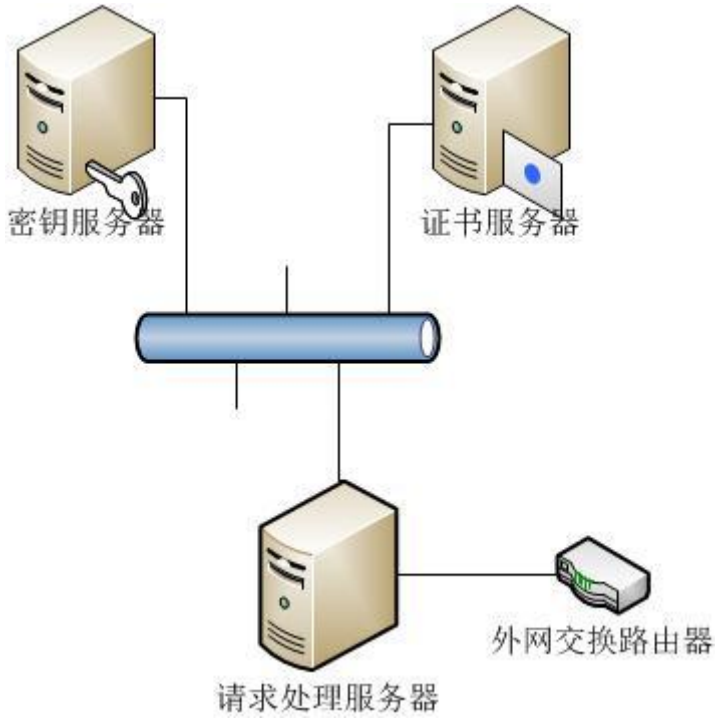

图3.1　实际系统模型

在实际系统实现之中，整个系统采用一台服务器来完成。生成密钥与证书采用 Linux 系统的 openssl 工具指令来生成。

生成 CA 证书的指令如下所示：

openssl req -text -x509 -newkey rsa:1024 -keyout cakey.pem -out cacert.pem

从-x509 参数，可以看出我们用我们采用 x509 格式来生成证书，rsa:1024 表示采用非对称加密算法 RSA 算法，密钥长度为 1024 位。-keyout 表示私钥输出，-out 表示公钥输出。

生成单位证书需要两个步骤，第一步生成证书，指令与生成 CA 证书类似。第二步为为 CA 签发证书，指令如下所示：

openssl ca -config ./openssl.cnf -in ${1}req.pem -days 10000

其中${1}表示需要签发的证书的名字，-days10000 表示证书的有效期。

在目前的系统中分发证书是通过加密邮件的形式

3.2.2 SSL 加密通信实现

当 CA 系统及证书向各个服务器发布之后，服务器双方便可以建立 SSL 通信[14]。

服务器端开启 SSL 通信的流程，同时列出了利用 C++编程需要调用的一系列 openssl[15]函数。

SSL 初始化操作

- SSL 库初始化 SSL_library_init();
- 载入 SSL 所有加密通信算法 OpenSSL_add_all_algorithms();
- 载入 SSL 错误集 SSL_load_error_strings();
- SSL 握手前环境准备 SSL_CTX_new(SSLv23_server_method())，在这里定义的是 SSLv2 和 v3 兼容的标准。生成 ssl_ctx 类型变量用于 SSL 握手前的环境准备。

通信前准备操作

- 载入需要通信单位的证书，这个证书里面包含 SSL 建立通信前需要与对方交换的公钥 SSL_CTX_use_certificate_file();
- 载入私钥，用来解密对方利用利用公钥加密的信息 SSL_CTX_use_PrivateKey_file();
- 私钥检验 SSL_CTX_check_private_key();
- 建立 socket 的标准过程，包括 socket(),bind(),listen()。

服务器接收客户端加密通信的过程

- 标准接收 accept 接收客户端的连接
- 根据 SSL_CTX 的内容建立 SSL 数据，用来握手与传输数据。
- SSL 设置 socket 管道 SSL_set_fd();
- 建立 SSL 连接 SSL_accept()，在这个过程中进行了 SSL 握手；
- 利用 SSL_read 与 SSL_write 进行加密数据收发
- 关闭 SSL 连接的步骤为关闭 SSL 连接 SSL_shutdown()，释放 SSL 数据 SSL_free(),关闭 socket close();

客户端向服务器发起 SSL 连接的过程

- SSL 初始化过程，与服务器 SSL 初始化相同。
- 建立 socket
- 利用 conncect()连接服务器
- 建立 ssl 变量

- 向服务器发起 SSL 连接 SSL_connect();
- 利用 SSL_read 与 SSL_write 进行加密数据收发

第4章 无极计算中间件通信系统开发

4.1 通信系统总体概述

无极计算中间件的通信系统已 Linux 的 socket 通信为基础，采用 C++语言进行开发。开发过程中采取同步通信与异步通信并存的方式，即无极计算中间件各层的服务器既可以采用同步通信的方式，也可以采用异步通信的方式。

具体开发过程中，同步通信与异步通信以类的形式进行封装。将所有有关通信的操作封装进同步通信类与异步通信类之中。其他各层以变量的形式引用同步通信类与异步通信类变量，通信操作以调用通信变量的成员函数来进行。

为了使通信类操作更加简便与面向对象化，设计通信基类，在通信基类之中以虚函数的形式定义了所有与通信有关的操作。同步通信类与异步通信类分别继承通信基类，并对各个通信的操作函数进行自己的实现。通信类的关系如下图所示。其中 ComBase 是通信类基类，CComSocket 是同步通信类，继承了 ComBase; CAsyncSocket 是异步通信类，继承了 ComBase。

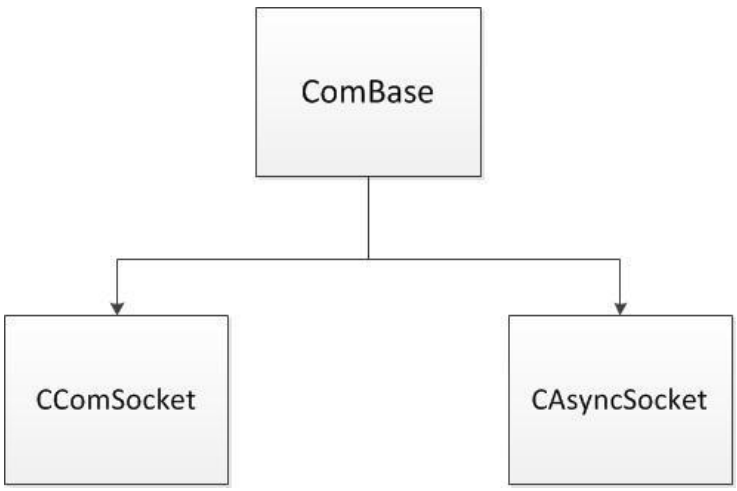

图4.1 通信类继承关系图

在 ComBase 定义了如下的操作，同步通信类与异步通信类则完成对如下操作的具体实现：

- CreateSock()创建 socket 端口
- Combind()绑定端口
- ComListen()开始 socket 监听
- ComAccept()接受对方通信
- ComConnect()向接收方发起连接
- ComRecv()接收对方发送数据
- ComSend()向接收方发送数据
- CloseSock();关闭 socket 通信，在异步通信中还要从事件队列里删除事件
- StartServer()开启服务器通信与监听，在异步通信中还有初始化事件队列的操作。

4.2 通信系统的实现

4.2.1 同步通信系统的实现

同步通信采用 Linux 标准的 C 的通信，socket 同步通信的流程如下所示，其中在 read 于 write 的过程之中，连接始终保持，发送与接收直到对方发送与接收完毕为止。

同步通信在处理并发的时候，接收每一个连接的时候新开启一个线程，函数调用流程如下所示：

pthread_create(&l_ThreadId,NULL,(void*(*)(void*))Process,(void *)this);

当结束完通信的时候要收回线程。

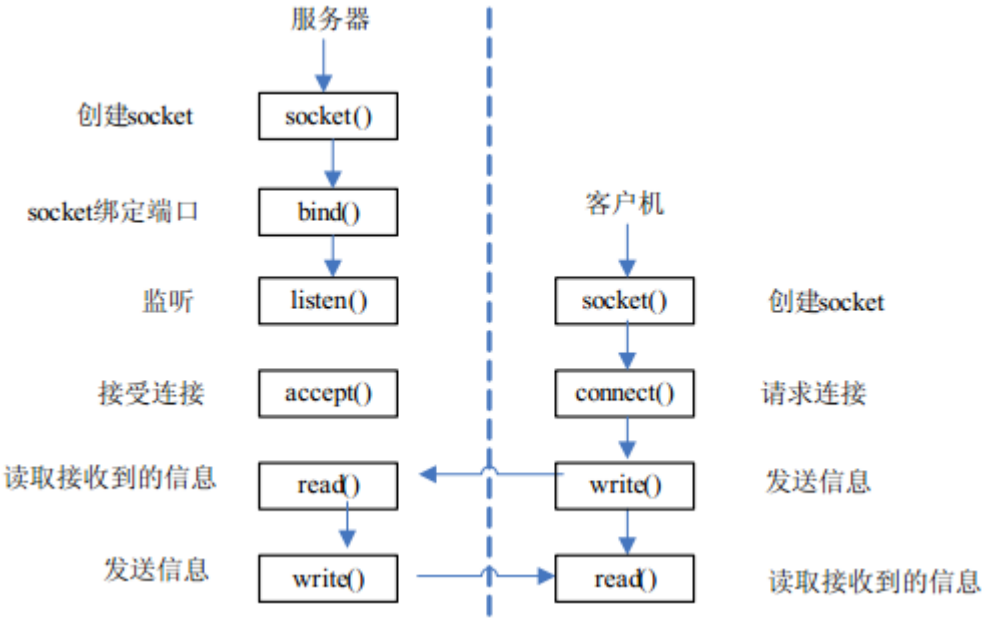

图4.2　同步通信流程图

4.2.2 异步通信系统的实现

本系统的关键是对异步通信的实现。在这里我们采用 Linux **错误!未找到引用源。**系统提供的 epoll 系列函数。

epoll **错误!未找到引用源。**是 Linux 内核为了处理大量句柄而为对 poll 的改进增强版，其基础是 Linux 的多路复用技术 select/poll。在本工程中，epoll 被用来处理高并发与异步通信，由于其对事件的边沿检测原理，能够有效地减小 CPU 占用率。通常单线程能够处理 15000 的并发数，其性能远优于同步通信的多线程 fork，与 select/poll 系类函数。

epoll 的高性能体现在其巧妙的事件结构上。对于传统的 select/poll 系统利用比较差，主要体现在每次调用都要重复的读取各个文件描述符的参数，不断地进行轮训。并且每次轮训结束之后都要删除 fd 队列，之后并重新建立。由于一般 fd 检测都是固定的事件，这样反复的建立与删除实在是浪费资源。epoll 本身采用的是事件通知机制，文件描述符的回调函数将其代理回到函数注册进队列。当产生新事件的时候，代理回调函数即唤醒文件进程及其回调函数，将其加入事件处理的 ready 队列，扫描量大大降低。

epoll 接口也十分简单，总共只有三个函数，epoll_create, epoll_ctl, 与 epoll_wait。

在无极计算中间件异步通信系统的具体实现，步骤如下

首先利用 epoll_create()创建一个 epoll 的句柄，启动服务器通信的过程其实与同步通信一致，但是在到系统开启监听 ComListen 的时候，为监听端口单独开一个 epoll 事件。

服务器开启正常的 socket 监听之后，便开始进入循环，利用 epoll_wait 函数来监听事件。系统主要监听 EPOLLIN 文件描述符可以读，EPOLLET 事件即将 EPOLL 设为边缘触发(Edge Triggered)模式，此种模式更加节省资源。但是会有可能产生操作符被挂起的错误。

比如通信当中经常要碰到如下的场景：

- 将文件描述符添加进 epoll
- 客户端向服务器发送 4K 的数据，数据已经在缓冲区
- 调用 epoll_wait，检测到数据写入事件
- 接收数据
- 继续调用 epoll_wait

由于是边沿触发，如果接收数据的时候没有把 4K 的数据接收完全，则系统不会触发新的事件。导致 epoll_wait 事件被挂起。所以需要改写 read 和 write 函数，使其可以读取缓冲区内部任意长度的数据，并且能够一次读完。当发送数据的时候我们将要发送数据的缓存长度加在所发送数据之前。当利用 read 函数接收的时候，首先将所发数据长度解析出来，其后根据解析出来的数据长度，将所有的数据一次读完。

每当系统接收到一个客户端发送的一个 connection 的时候，服务器在 accept 这个连接之外，就利用 epoll_ctl 增加一个新的监听事件。

在结束异步通信的时候，不仅需要 close()函数来关闭端口，还需要从事件队列中删除 epoll 队列中删除对于该文件描述符的监听。

第5章 基于通信与安全系统的无极计算中间件系统实现

5.1 通信系统与安全系统的整合

5.1.1 通信类与加密通信的整合

无极计算对于安全通信的设计是基于 SSL 通信协议，无极计算的通信基础是基于 socket 通信的同步通信与异步通信。本系统的目标就是可以在无极计算通信之中既可以同步计算与异步计算，既可以加密通信与非加密通信。在实际编译过程中，可以通过更改编译选项来进行同步异步，加密与非加密的选择。通信类与安全通信类的关系设计如下图所示：

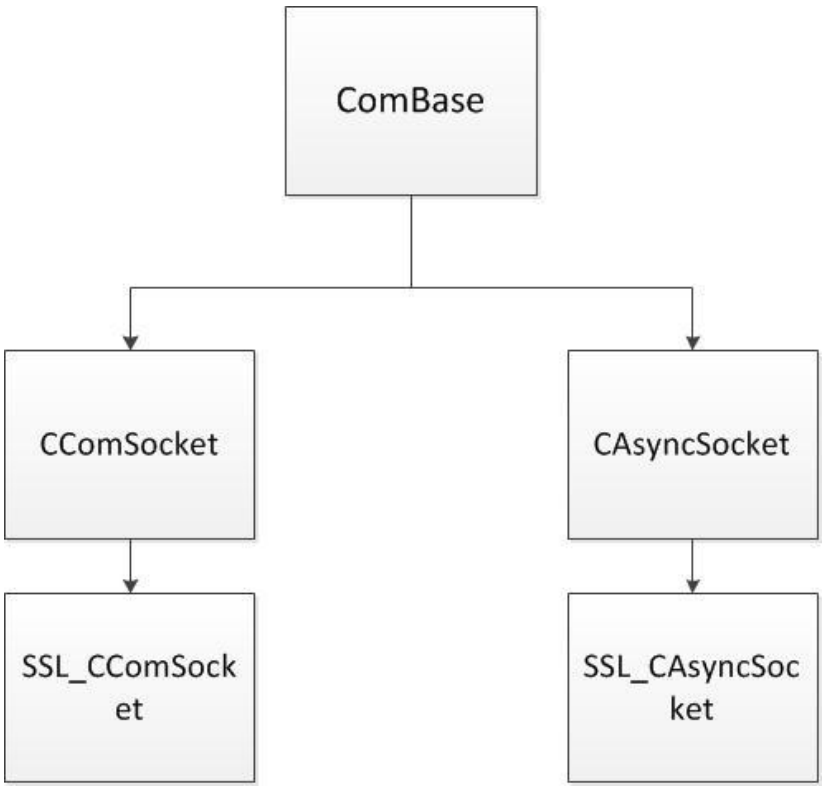

图5.1 通信类与安全通信类继承关系图

如上图，其中 ComBase 是通信类的基类，CComSocket 是同步通信类，CAysncSocket 是异步通信类，继承 ComBase。SSL_CComSocket 继承于同步通信类，只是增加了 openssl 中的操作函数。对通信操作的各个流程进行了改造。

对于安全通信的服务器开启操作，只需要增加 SSL 初始化的一系列操作，包括，SSL 库初始化 SSL_library_init();载入 SSL 所有加密通信算法 OpenSSL_add_all_algorithms();载入 SSL 错误集 SSL_load_error_strings();SSL 握手前环境准备 SSL_CTX_new(SSLv23_server_method())，在这里定义的是 SSLv2 和 v3 兼容的标准。生成 ssl_ctx 类型变量用于 SSL 握手前的环境准备。

对于 ComAccept 操作，只要将 Accept 函数换成 openssl 的 SSL_Accept。对于 ComConnect 操作，只要将 Connect 函数换成 openssl 的 SSL_Connect 函数。对于传输数据与接收数据，类似地只要把 Linux C 语言网络通信函数换成相应的 SSL 通信函数即可。

在结束通信的时候只需要在利用 close 关闭 socket 之后，利用 SSL_free 释放 SSL 数据类型。

5.1.2 通信系统与安全系统同无极计算整体系统的整合

在无极计算的各个服务器程序中，包括元素服务器，逻辑调度服务器，CI 服务器，虚拟组织服务器，与流程服务器中，全部都涉及到加密通信操作。根据面向对象的设计原则，应当将所有与通信有关的操作都封装进安全与通信类之中。其他服务器程序只是像工具一样的调用通信与加密通信的操作。

具体实现就是，在服务器程序类中增加通信与加密通信的变量，通过调用该变量的成员函数来进行与之有关的操作。

5.2 无极计算中间件系统的编译管理

在实际无极计算中间件开发的过程当中，需要检查系统的环境变量、文件是否存在等问题，需要链接大量其它的链接库，需要为程序的使用者根据自己的需要增加编译与安装选项。本系统中采用 Autotools 系列编译工具为整个程序进行编译与安装管理。需要打包者做得主要工作就是编写 configure.ac 与 Makefile.am。

5.2.1 configure.ac

configure.ac 文件主要是用来检查系统的环境，检测需要用到的库文件与程序，定义整个系统的编译结构。一般来说，可以通过 autoscan 命令生成 configure.ac 的模板。该指令可以将我们编写的源代码中所有涉及的函数，头文件检测出来，并写进 configure.ac 待 autoconf 来查找。

在我们的系统之中，需要检查系统是否安装了 openssl 的相关链接库与头文件。其中对于加密通信来说，需要系统有 libcryto.so 与 libssl.so 两个链接库。在 configure.ac 中采用 AC_CHECK_LIB([crypto], [rand])来检查是否有 libcrypto 链接库，采用 AC_CHECK_LIB([ssl], [SSL_load_error_strings])来检查是否有 libssl.so。如果检查到存在该链接库，autotools 工具会自动将该链接库引入到个子程序的 Makfile 之中。

在编译与 SSL 通信有关的程序中，需要引入 openssl 的头文件。为了方便编译，我们在这里定义变量 INCLUDEFLAGS，其中定义了头文件搜索位置。利用 AC_SUBST 可以讲定义的头文件位置变量传递到个子程序的 Makefile 之中。

为了能够让程序使用者与无极计算二次开发者能够根据自己的需要开发程序，需要添加自己的选项。在 configure.ac 中，先定义变量 USERDEF_INSTALL，通过判断用户对于该变量的定义，利用 AC_DEFUN 这个宏来进行判断，根据判断所得，将需要安装的子程序文件夹通过 AC_SUBST 以变量的形式传到子程序的 Makefile.am 当中。

对于系统环境的检测，最关键的是判断系统时 32 位系统还是 64 为系统，在这里，我们采用 test $SYSTEMTYPE -eq 32 即可判定系统是不是 32 位系统。

5.2.2 Makefile.am

针对每一个子程序的 Makefile，需要编写 Makefile.am 模板。利用 Makefile.am 实现 Makefile 非常简单，只需要添加程序名字，需要涉及编译的源文件，需要的链接库以及必要的编译选项即可。以元素服务器的基类程序为例

```
bin_PROGRAMS=EleServer
EleServer_SOURCES=EleServer.cpp CElement.cpp
EleServer_LDADD=$(top_srcdir)/src/ele/libElement.la
$(top_srcdir)/src/common/readconf/libreadcfg.la
```

$(top_srcdir)/src/common/comdb/libRecordlist.la

$(top_srcdir)/src/common/verifycom/libVerify.la

EleServer_CPPFLAGS=-g -I$(top_srcdir)/src/include $(INCLUDEFLAGS) -DWITH_DB

SUBDIRS = . host data display soft

其中 bin_PROGRAMS 代表需要编译程序的名称，XXX_SOURCES 代表编译该程序所需要的头文件，XXX_LDADD 代表编译该程序所需要的链接库，XXX_CPPFLAGS 代表编译该程序的时候所需要的编译指令，SUBDIRS 代表需要编译的子文件夹。通过更改 SUBDIRS 变量，我们可以根据用户需求来编译需要编译的子程序。

在编译无极计算中间件过程之中，不仅需要可执行程序，还需要编译链接库。Autotools 为我们提供了非常好的编译工具，即 Libtool。Libtool 将编译动态链接库的复杂操作都隐藏了起来，解决了繁琐复杂的链接库依赖性问题。利用 Libtool 可以帮助开发者减少大量的编译链接库的细节问题。其关键技术在于编译过程中生成的共享库.la 文件。共享库其实是一个文本文件，其中主要定义了和各个链接库的依赖关系。如果希望得到传统的链接库文件例如.so 动态链接库或者.a 静态链接库，只需要通过 make install 指令，即可以通过单纯的共享库.la 文件，即生成静态链接库文件与动态链接库文件。本项目中，希望将对于元素的操作都封装进一个动态链接库之中，所以在这里我们就编译了元素基类的动态链接库，如下：

lib_LTLIBRARIES = libElement.la

libElement_la_SOURCES=CElement.cpp

libElement_la_LDFLAGS=-c $(INCLUDEFLAGS) -version-info 0:0:0

lib_LTLIBRARIES 即可以指定需要编译的共享库的名称，XXX_SOURCES 指定需要涉及编译的文件，libElement_la_LDFLAGS 可以增加编译选项，在这里我们指定了头文件的搜索路径，链接库的版本号。此外我们还可以通过 XXX_LIBADD 增加需要编译的其它需要依赖的链接库。编译过程十分方便，可以让开发者更加专注于程序开发的本身工作上。

5.2.3 无极计算中间件整体编译流程

为了简化程序使用者使用源代码的使用流程，将所有编译流程采用脚本的形式整合起来。一般利用 Autotools 的编译流程如下：

- autoreconf –i
- ./configure
- make
- make install

configure 脚本有 prefix 参数可以指定程序安装位置，需要安装脚本可以传递变量参数。所以脚本的可选参数应该设计，使得一方面能够满足使用者选择编译安装的子程序，另一方面可以传入编译指令需要的所有参数。

由于本程序还需要建立数据库才可以进行使用，所以脚本中增加了建立数据库需要的指令。用户可以根据自己装的子程序，选择需要安装的数据库。

5.3 无极计算系统框架与执行示例

5.3.1 无极计算中间件系统框架

无极计算中间件按照无极计算架构理念设计，以安全系统与通信系统为基础，建立了分层通信的一系列服务器程序。无极计算中间件的架构与流程图如下所示，其中所有通信都是基于 SSL 的加密通信，各层服务器程序直接调用加密通信的动态链接库。

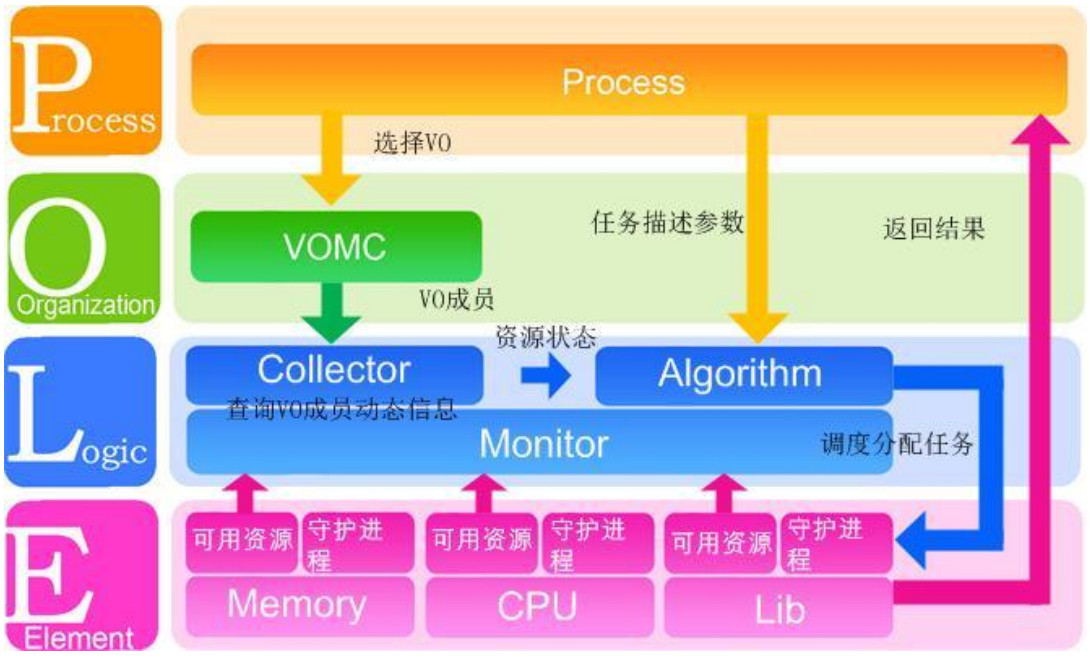

图5.2　无极计算中间件架构与流程图图

对于流程层，有基于流程管理的服务器，其主要功能为接收用户对于流程的设计并将其分配到组织层核查权限，分配到逻辑层执行调度。

组织层分别运行 CI 证书认证服务器与 VO 虚拟组织服务器，CI 服务器对系统内所有单位进行登记注册与权限管理，VO 虚拟组织服务器分别管理虚拟组织内部各个元素与用户的权限。

逻辑层可以运行多种逻辑服务器，逻辑服务器可以根据其功能与算法进行多种划分。对于二次开发者，逻辑服务器程序提供调度算法开发接口，可以根据需要采用任意的逻辑调度算法来调度元素。

在元素层根据各个元素不同功能进行划分。其中最重要的是对于主机的封装，在这里我们既可以对物理机进行封装，也可以对虚拟机进行封装。对虚拟机封装体现了云计算虚拟化的特点，也增强了无极计算计算资源的动态调度能力。无极计算系统支持 Xen、KVM 与 Virutalbox，并能对其进行统一调度，迁移与负载均衡。无极计算还正在开发对于文件系统与存储的元素封装。对于二次开发者，提供了元素基类接口，方便其对于其它元素的封装。

无极计算的特点在于系统可扩展性、任务分工的明确性与资源共享的安全性。分散性体现在系统结构是松耦合的，将原有的计算机系统重新划分成不同的层次，便于服务重组。而任务分工的明确性指在无极计算的四层体系构架下，层与层之间的任务分工明确，任务的执行过程是流水作业的，提高系统的效率。而资源共享的安全性指通过构建虚拟组织，引入证书认证机制，使得组织内成员可以安全的共享资源。

5.3.2 无极计算中间件的运行环境与开发环境

(1) 运行环境

服务器：HP ProLiant DL580 G5

CPU 数：12 核×2

主　板：AMD 主板芯片组

内　存：4G×4

硬　盘：SATA 1000GB 7200 转

操作系统：Linux CentOS 5.5

依赖软件：OpenSSL 1.0

Mysql

Unixodbc

(2) 开发环境

笔记本：联想 Y460

CPU 数：2 核

内　存：3G

硬　盘：500G

操作系统：Linux CentOS 5.5

5.3.3 无极计算系统的执行流程

用户具体执行操作的步骤如下：

1、用户选择执行任务的 VO

2、用户将任务提交给逻辑层，任务包含用户证书，逻辑层维护一个任务队列

3、逻辑层向组织层发出申请，查询用户在该 VO 中的权限

4、VO 服务器将用户权限信息发送给逻辑层

5、逻辑层接受 VO 服务器返回的查询结果，如果用户拥有该 VO 的执行权限，则逻辑层向组织层发出申请，查询该 VO 拥有执行权限的元素列表

6、逻辑层接收该 VO 的元素列表，同时查询逻辑层维护的元素状态信息，调度器根据元素列表及元素状态匹配执行任务的元素节点

7、逻辑层将任务发送给执行任务的元素节点

8、元素节点接收任务执行任务后将结果发回给用户

以上任何一个连接过程均利用数字证书进行身份认证，每一次指令发送均利用证书进行加密。用户的所有操作与流程管理都可以在如下界面运行。而提供无极计算的服务器需要先建立相应的数据库之后，直接运行安装好的程序即可。

图5.3　无极计算系统操作主界面

第6章 无极计算系统整体评估及未来工作展望

6.1 无极计算整体评估

通过对于无极计算安全通信系统的研究，最后采取 SSL 安全通信方案，该方案整体上满足了无极计算系统对于加密通信的需求，同时也是目前互联网上所采用的加密通信的主流方法，方便同互联网其它设备与服务进行安全通信方面的兼容。

在无极计算异步通信机制，最终采用以 Linux 内核的 epoll 为基础的并发解决方案。该方案经系统试验可以达到单线程 15000 的并发访问量量，基本满足了无极计算系统的初步要求，但在未来产品升级过程中， 还应该优化并发方案。

整个无极计算系统采用 C++来进行开发，虽然在开发效率方面偏低，但是在程序执行效率方面较高，满足了科研中对于高性能计算的系统需求。

无极计算 ELOP 云计算中间件国内尚无同类产品，而国际上关于云计算体系构架如何实现也没有标准的模式，目前也只有谷歌（Google）、IBM、亚马逊（Amazon）推出了自己设计的云计算解决方案，无极计算中间件具有科学研究的意义。从产业化角度考虑，云计算作为未来计算系统的一个发展趋势，本项目具有深远的市场前景。一方面，因为无极计算通过将资源进行虚拟化提供给用户，终端用户本身不需要实际的计算和存储设备，只需要简单的输入输出终端接入数据中心和计算中心，这将大大压缩成本。另一方面，由于用户终端并无存储设备，这将降低企业知识库失窃的可能性。这将提高企业知识产权的安全性。考虑从计算资源利用率的角度考虑，无极计算软件综合调度资源池的物理资源，能够最大可能的实现资源的高效分配与利用，有助于减少能耗，实现绿色计算。无极计算在未来的商业及科学计算领域必将得到广泛应用。

6.2 未来工作展望

目前无极计算中间件还在处于与产品研发阶段，安全系统与加密通信以确定解决方案，各层服务器程序也已有初步的示例程序。但是程序，在逻辑层调度算法接口，流程层程序设计方面还不完善，在权限管理与系统界面接口方面也只是初步的示例级别的程序，离产品级别还比较远。

在未来的开发工作中，主要还有以下开发工作需要重点进行

- 对于逻辑层进行进一步抽象，合理设计调度算法开发接口
- 在流程层引入业务流程管理机制（Business Process Management 简称 BPM），选择其核心支持无极计算的功能进行借鉴开发
- 合理设计无极计算系统中各个单位的权限关系
- 研究界面系统的 php 与 C++程序比较好的兼容解决方案。
- 加强系统虚拟化的研究，实现 XEN，KVM，Virtualbox 的统一调度。

插图索引

参考文献

[1] Kai Hwang, Geoffrey C. Fox, Jack J. Dongarra, Distributed and Cloud Computing From Parallel Processing to the Internet of Things. U.S.: Elsevier, 2010

[2] Foster, I., Kesselman, C. Computational Grids. Morgan Kaufmann, 1998.

[3] http://www.utilitycomputing.com

[4] Mark Baker. Cluster Computing White Paper, 2000.12.

[5] http://en.wikipedia.org/wiki/Cloud_computing

[6] D. E. Atkins, et al., Revolutionizing Science and Engineering through Cyberinfrastructure. National Science Foundation Blue – Ribbon Advisory Panel on Cyberinfrastructure, January 2003.

[7] J. Cao (Ed.), *Cyberinfrastructure Technologies and Applications*, Nova Science Publishers, 2009.

[8] 曹军威. 赛百平台及其技术挑战. 国际学术动态,，2010 年 2 期

[9] MIT Infrastructure Science and Technology Group. http://web.mit.edu/istgroup/.

[10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, et, al, "Above the clouds: A Berkeley View of Cloud Computing", *Technical Report No. UCB/EECS-2009-28, University of California, Berkerley*, 2009.

[11] F. Zhang, J. Cao, C. Hong, J. J. Mulcahy, and C. Wu, "Provisioning Virtual Resources Adaptively in Elastic Compute Cloud Platforms", *Int. J. Web Services Research*, 8(3), 54-69, 2011.

[12] 刘鹏.《云计算》.北京：电子工业，2011 年 5 月. 8~20

[13] Enabling Distributed Computing Systems with elop. J. Cao and S. Chen. Proc. 3rd Int. Conf. on Networking and Distributed Computing, Hangzhou, China, 2012.

[14] 王志海. OpenSSL 与网络信息安全. 童新海, 沈寒辉. 北京：清华大学出版社, 2007

[15] OpenSSL 官方网站 http://www.openssl.org

[16] 康弗瑞, kang fu rui S. (Convery, Sean). 网络安全体系结构. 北京 ： 人民邮电出版社, 2005

[17] 白辉. 在线收全平台的设计与实现[D]. 广州: 华中理工大学软件学院，2010: 5-10.

[18] Transport Layer Security, http://en.wikipedia.org/wiki/ Transport_Layer_Security

[19] John Calcote, Autotools: A Practioner's Guide to GNU Autoconf, Automake, and Libtool

[20] GNU General Public License Version 3, 29 June 2007, Copyright © 2007 Free Software Foundation, Inc.

致　　谢

本毕设是同无极计算中间件开发项目协同进行。在项目进行中，我的指导老师曹军威研究员同我进行了多次深入的讨论，从研究方向上给予指引，从研究工作方法上给予悉心指导，在软件开发工作上给予纠正，在科研工作方面给予鼓励，从而使我能够顺利地完成毕设，在这里首先要感谢曹老师的帮助。

实验室的万宇鑫师兄在我的具体研究工作方面给予了耐心的解答，亲身的帮助。没有他我也就不能这么快的了解无极计算的系统架构，也不能在软件开发方面有这么大的进步。同时也要感谢张家港电力研究院物联网所的各位同仁，没有他们的支持，软件开发工作也不能进展的如此顺利。

声 明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____　　日　期：_____

附录 A 外文资料翻译工作

针对本毕设项目课题范围，我选择了并行计算介绍方面的资料进行翻译。具体为黄凯教授“Distributed and Cloud Computing From Parallel Processing to the Internet of Things”第二章的前半部分进行翻译。现附如下，其中的章节号代表了原书中的章节号。

2. 可扩展并行计算的机群系统

总结

机群系统使可扩展分布式并行计算系统在科学与商业领域都成为可能。本章旨在阐释如何建立集群结构的大规模并行处理器。我们关注硬件、软件、中间件、操作系统的设计与实现以达到可扩展性、可编程、单一系统映像以及容错能力。我们将研究 Tianhe-1A、Cray XT5 Jaguar 和 IBM Roadrunner 的集群结构。研究也包括 LSF 中间件和 Linux 集群里的任务与资源管理系统 MOSIX/OS、GPU 集群以及集群扩展的网格和云。本章之研究物理层面的集群。虚拟机群将在第 3、4 章讨论。

2.1.大型并行计算集群

计算机机群是多个互相连接的独立电脑的集合，其能够作为一个独立的计算资源池进行协同工作。机群系统在业务层上高度并行，通过独立操作来实现高可用性（high availability）。机群系统与大规模并行处理器（MPPs）的优点在于可扩展性、高可用性、容错性、构建生长以及商业元件的可用性。这些特点支撑了硬件、软件与网络元件的升级换代。集群计算作为主流框架在上世纪 90 年代中期十分流行。已证明向量超级电脑在高性能计算方面更加节约资源。

2010 年计算能力前 500 名的超级电脑中，85%的机群系统与大规模并行处理器是建立在同质节点之上的。机群系统是当今超级电脑，计算网格以及网络云的基础。我们已经对计算机产生了依赖。根据互联网数据中心（IDC）预测，高性能计算（HPC）市场规模将从 2010 年的 85 亿美元上涨至 2013 年的 105 亿美元。前 500 名的超级电脑中大多数被用在科学与工程中的高性能计算应用。同时，高吞吐计算（high throughout computing）的服务机群在商业与网络服务方面的应用也在高速增长。

2.1.1. 机群发展的趋势

机群系统的支持架构从过去的互相连接的高端大型机发展为大量的 X86 引擎。机群系统开始是大型机的互相连接，例如 IBM Sysplex 和 SGI Origin 3000。最开始，机群系统是为了满足协同计算以及企业应用中的高可用性而产生。后来，机群系统向小型机网络发展，例如 DEC 的 VMS 系统，大量的 VAX 互相连接并共同应用相同的磁带控制器。Tandem 的 Himalaya 被设计为容错的联机事务处理商用机群。

在上世纪 90 年代早期，新的发展是以 Unix 为基础的工作站的建立，典型代表为 Berkeley NOW（Network of Workstation）和 IBM SP2 AIX 服务器集群。2000 年之后，发展趋势为 RISC 或者 x86 引擎集群。集群化的产品现在有集成系统、软件工具、可用公共设施以及操作系统扩展。集群的发展趋势正好契合了计算机工业小型化的趋势。更小节点的支持机群可以通过模块化的增加来增加销售额。从 IBM、DEC、Sun、和 SGI 到 Compaq 和 Dell，计算机产业增加了的低成本服务器或者 x86 计算机的机群产量，由于其低消耗、可扩展性以及高可用性的特点。

2.1.1.1. 机群系统的里程碑

机群系统是计算机结构研究的热门领域。高速通信、任务调度、单系统映像、高可用性都是机群系统研究中的重要方向。表 2.1 列出了一些里程碑式的机群系统研究工作于商业产品。这些过去的系统的详细信息可以在文献[14]中找到。这些里程碑式的项目是过去 20 年中集群硬件与中间件发展的前沿。每个列出项目的有其独特的特点。现代机群系统中的高性能计算集群在 2.5 节中进行介绍。

NOW 项目侧重机群系统的整体问题，包括结构、网络服务器的软件支持、单映像系统、I/O 与文件系统、高效通信、高可用性。Rice University TreadMarks 是软件应用于内存共享工作站集群的典型代表。有研究建立 Sun Soloris 系统工作站。一些机群系统正在发展，但还未得到成功的商业应用。

SMP 服务器集群的 Unix 系统的机群系统在 VMS/OS 上运行扩展，主要用在高可用性的应用当中。AIX 机群系统 Power2 节点与 Omega 网络之上并被 IBM Loadleveler 和 MPI 扩展支持。可扩展，容错的用作联机事务处理、数据库运作的机群系统，该系统建立在不停止的操作系统之上。Google Search Engine 尽力在 Google 的商业组件之上。MOSIX 基于 Linux 系统机群，网格与云上的分布式操作系统，组开始由 Hebrew 大学在 1999 年开发。

项目	机群系统特点
DEC VAXcluster(1991)	对称处理器的 UNIX 服务器集群，运行 VMS OS 来进行扩展，主要终于高可用性的应用中
U.C. Berkeley NOW 项目(。1995)	无服务器的工作站网络，用来进行 active messaging、协同备份与 GLUnix 的开发
Rice University TreadMarks(1996)	软件应用共享内存的 UNIX 集群基于页面迁移
Sun Solaris MC Cluster(1995）	建立在 Sun Solari 的工作站集群；一些操作系统的功能被开发出来但未取得商业上的成功
Tandem Himalaya Cluster(1994)	可扩展，容错的用作联机事务处理、数据库运作的机群系统，该系统建立在不停止的操作系统之上
IBM SP2 Server Cluster(1996)	建立在 Power2 节点与 Omega 网络上的 AIX 服务器，被 IBM LoadLeveler 和 MPI 扩展支持
Google Search Engine Cluster(2003)	4000 个节点的服务器集群包含文件系统并有容错能力，用作互联网搜索与网络服务应用。
MOSIX(2010) www.mosix.org	Linux 系统机群，网格与云上的分布式操作系统，目前在研究领域中

表 2.1

2.1.2. 机群系统的设计目标

机群系统根据以多种方式进行分类。我们通过六个指标对其进行分类：可扩展性、封装、可控性、同一性、可编程性和安全性。

2.1.2.1. 可扩展性

机群系统是以构建生长的概念为基础。从数百个同一处理器的集成到一万多个多核节点的高度集成不是一个简单的任务。可扩展性受到一些因素限制，比如多核芯片的技术水平、集成拓扑、集成方式、能耗以及散热措施。我们的目标是克服前述的因素以实现更好的扩展能力。我们还要考虑其他的限制因素，比如存储墙、磁盘 I/O 瓶颈、容许时延等。

2.1.2.2. 集成方式

机群节点可以以紧密或者松散的方式进行集成。在一个紧密的机群里，节点被放置在一个屋子里的一个或者多个机箱中，节点没有连接外设（显示器、键盘、鼠标等）。在一个松散系统里，节点通常连接外设（比如完整的 SMP，工作站或者 PC），被放置在不同的屋子里面，不同的楼里甚至相距更远。集成方式直接影响连接线的长度，以及连接技术选择。紧密集成的系统高带宽，低时延，松散系统通常通过局域网或者广域网进行连接。

2.1.2.3. 控制方式

机群系统可以按控制与管理方式分为集中控制与分散控制。紧密集成的系统通常是集中控制，松散集成的系统通常采用分散控制。在集中控制的系统中，所有的节点呗同一个中央处理器管理与控制。在分散系统中，节点有多个管理者。例如，设想一个集成一个部门的多个工作站的系统，每个工作站分别属于不同的雇员。所有者在任何时候可以重新配置、升级、关闭工作站。没有单一节点进行控制使系统的管理非常困难。它还要求集成电镀、负载均衡、检测、计算等一些特殊的技术。

2.1.2.4. 同一性

同一的机群系统采用同一平台的节点，采用相同的处理器结构以及相同的操作系统，通常都是从同一个卖家买的。非同一系统通常采用不同的平台。互通性是非同一系统的重要关注点。例如进程迁移在负载均衡与可用性里经常采用。在同一系统中，二进制进程映像可以迁移到其他节点中继续运行。在非同一系统中就很不方便，如果二进制代码在进程迁移的过程中在其它节点中无法运行。

2.1.2.5. 安全性

机群内部通信可以分为公开的和封闭的两种。在公开机群中，节点间通信通道向外界公开。外部可以访问通信通道，节点间采用标准的通信协议。这种机群比较容易实现，但是有如下缺点：

公开的状态下，机群间的通信会不安全，除非有通信子系统有额外的功能来保证隐私性和安全性。

外界通信在不确定的情况下会干扰机群间通信。例如，BBS 的网络拥堵可能会干扰生产作业。

标准通信协议通常协议栈比较高。

在封闭系统中，机群间通信通常与外界隔绝，这样就可以减轻前述的问题。但缺点是目前没有有效的标准机群内部通信。因此，大多数商业或学术机群系统通过一种类似标注的协议来实现高速通信。

2.1.2.6. 专用系统与企业系统

专用机群一般安装在机房之中的大量机柜之中。它们一般被设置成同类的计算机节点，并有专门的管理组。专用机群一定程度上替代了传统计算机架构以及超级电脑。机群可以被当做单独的电脑来控制。许多用户都可以登录机群来进行批量计算。机群可以提高计算能力，减小运行时间。

企业机群通常主要被用作单独资源节点。每个节点通常为多核处理器，工作站，或者为 PC 以及所有的周边设备。这些节点通常在地理上是分布式的，不一定被安置在同一个地方。节点通常被多用户所共有。机群管理节点通常只有对子节点的有限权限，并且节点可以所有者在任何时间关闭。所有者私人任务的优先级高于机构任务。机群内的节点通常是异构的。节点一般通过以太网的形式连接起来。大多数数据节点通常为廉价的服务器。虚拟节点在升级数据中心之中起到了很大的作用。我们将在第 6 章讨论虚拟机群，在 7、8、9 章讨论云。

2.1.3. 机群重要的设计要素

在本节之中，我们将对机群以及大规模信息处理机进行分类。之后我们将阐述机群以及大规模信息处理时系统的设计原则，包括物理以及虚拟机群。这些系统通常在计算网格与国家实验室、商用数据、超级计算机以及云之中。对于大规模信息处理系统更好的理解可以帮助我们更好地理解大规模网格系统的输入输出以及互联网中的云。在设计网格与云的时候，一些重要的原则必须被考虑进去。尽管在这个领域已经有了大量的工作，但是还是有很多值得研究与挖掘的领域。

2.1.3.1. 扩展性

扩展性意味着资源的规模扩展的时候，性能会按比例增强，包括计算节点、内存、I\O 带宽等。当然，增加规模与减小规模都需要这样，根据具体应用需求以及效率成本核算。我们不应该在设计机群以及大规模信息处理系统的时候忽略可扩展性。

2.1.3.2. 单系统映像

简单地被以太网连接起来的工作站群不能被称作机群。机群是一个单一系统。假设某一工作站有 3G 次运算每秒的计算能力，512M 的内存，5G 的硬盘

存储，可以支持 50 个用户同时在线工作以及 1000 个线程。当将 100 个这样的工作站组成一个机群的时候，我们相当于在使用一个超大的工作站，拥有 300G 次每秒的运算能力，50G 的内存，500G 的内存，同时支持 5000 个用户同时在线工作以及 100000 个线程同时运行。这是一个很明显的目标，但是非常难实现，单系统映像技术的目标就是解决这一问题。

2.1.3.3. 可用性支持

机群需要提供成本合理的高可用性支持，因此需要处理器、内存、硬盘、I/O 设别、网络和操作系统映像有大量的空余能力。然而为了实现这一目标，则需要一定的可用性技术。我们将在本书的后面讨论 DEC 机群与 IBM SP2 的时候阐述这些技术，研究它们是如何实现高可用性的。

2.1.3.4. 机群任务管理

机群希望比传统的工作站与 PC 达到更高的系统使用率。任务管理软件提供了批处理，负载均衡，并行处理和其它功能。我们将在机群任务 3.4 节调度系统中研究这些问题。我们需要一些专用的软件来处理并行任务处理的问题。

2.1.3.5. 节点间通信

由于计算节点的高度复杂性，机群节点不能像 MPP 节点那样紧密的封装。机群节点件的物理连接线比 MPP 节点要长。对于中心机群更是这样。更长的连接线意味着更长的延时。但更重要的是，更长的连接线会导致稳定性更差，时钟时序不调，以及异步通信。这些问题需要一个更加安全可靠的通信协议来解决，通常会使协议栈更高。机群通常应用一般网络，比如以太网，利用标准协议，例如 TCP/IP。

2.1.3.6. 容错与恢复

机群通常可以容忍单独节点错误。通过冗余，机群通常可以容忍一定程度的错误。心跳机制通常可以帮助检测节点的运行状态。当一个节点发生错误时，运行在出错节点的任务可以移动到救援节点上执行。回滚机制可以帮助系统恢复到保存节点。

2.1.3.7. 机群家族分类

计算集群 这种机群通常被设计适应一个大规模的计算任务。一个很好的例子是天气预报的数值仿真。计算集群处理很少的 I/O 任务。当一个计算任务需要集群内的大量通信的时候，机群需要共享一个专用网络，因此这些节点通常是同质的且是紧耦合的。这种机群通常被称作 Beowulf 机群。

当通信仅限在一小部分高任务节点时，它们通常被称作计算网格。紧耦合的计算机群通常用来解决超大规模的计算问题。在计算机群上通常运行着 MPI 或者 PVM 中间件。

高可用性机群 高可用性机群通常具有高容错性来支持高可用性的服务。高可用性机群拥有大量的冗余节点来支持容错。最简单的高可用性机群有两个节点，并能够互相恢复错误。当然更多的冗余则可用性也更高。高可用性机群应该避免所有单节点出错。很多商用的高可用性机群支持多样的操作系统

负载均衡机群 这种机群由于采用了负载均衡策略，所以拥有更高的系统利用率。所有节点以虚拟机的形式进行作业。这种需求是由大量分散在机群中的用户产生的。因此需要进行负载均衡，以提高系统的使用效率。系统需要中间件以支持负载均衡与任务迁移。

2.1.4. 世界前 500 位超级电脑分析

每六个月，Linpack 便对世界前 500 名的超级电脑进行排名。排名标准是一组大数据的程序。排名像比赛一样每年都会更新。本节中，我们将会通过历史数据在时间的跨度上对他们的架构、运行速度、操作系统、国籍、应用进行分析。此外我们还将比较 2010 年排名前五名的超级机。

2.1.4.1. 架构演化

我们很有兴趣去如图 2.1 一样观察世界前 500 名的系统架构。在 1993 年，有 250 台的架构是 SMP 架构，但到了 2002 年 6 月所有 SMP 架构的电脑都消失了。大多数 SMP 架构的电脑都采用共享内存与共享 I/O 设别。1993 年，有 120 电脑是 MPP 架构。在 2005 年左右上升到了峰值，大约 350 台。但 2010 年下降到不到 100 台。SIMD 结构的及其在 1997 年就消失了。1999 年，只有少数几台采用机群结构。但现在前 500 名中大约有超过 400 台都采用这种结构。

2010 年，前 500 名中大约有 420 台采用机群结构，80 台采用 MPP 结构。这两种结构最主要的区别为系统元件不同。机群系统通常采用普通的硬件、软件和网络元件。MPP 通常采用定制的计算节点，主板，调制解调器和机箱来连接各个节点。MPP 需要高带宽，短延迟，更高的能效以及高可靠性。机群由于可以进行模式扩展因此也更经济。事实上，正是因为 MPP 更贵，所以它们数量更少。一般来说，每个国家一般只有很少几台的 MPP 架构机。

2.1.4.2. 运行速度

图 2.2 表明了世界前 500 名电脑从 1993 年到 2010 年的运行速度。y 轴表示以 Gflops，Tflops 和 Pflops 为单位的运行速度。中间的线表示 17 年中最快

的电脑的运行速度。最高速度从 58.7Gflops 上升到了 2.566Pflops。下面的曲线表示了第 500 名电脑的运行速度，从 1993 年的 0.42Gflops 上升到了 2010 年的 31.1Tflops。上面的曲线表示前 500 名电脑运行速度的总和，2010 年所有前 500 名电脑运行速度的总和为 43.7Pflops。从图中我们还发现这些电脑的运行速度几乎是线性增长的。