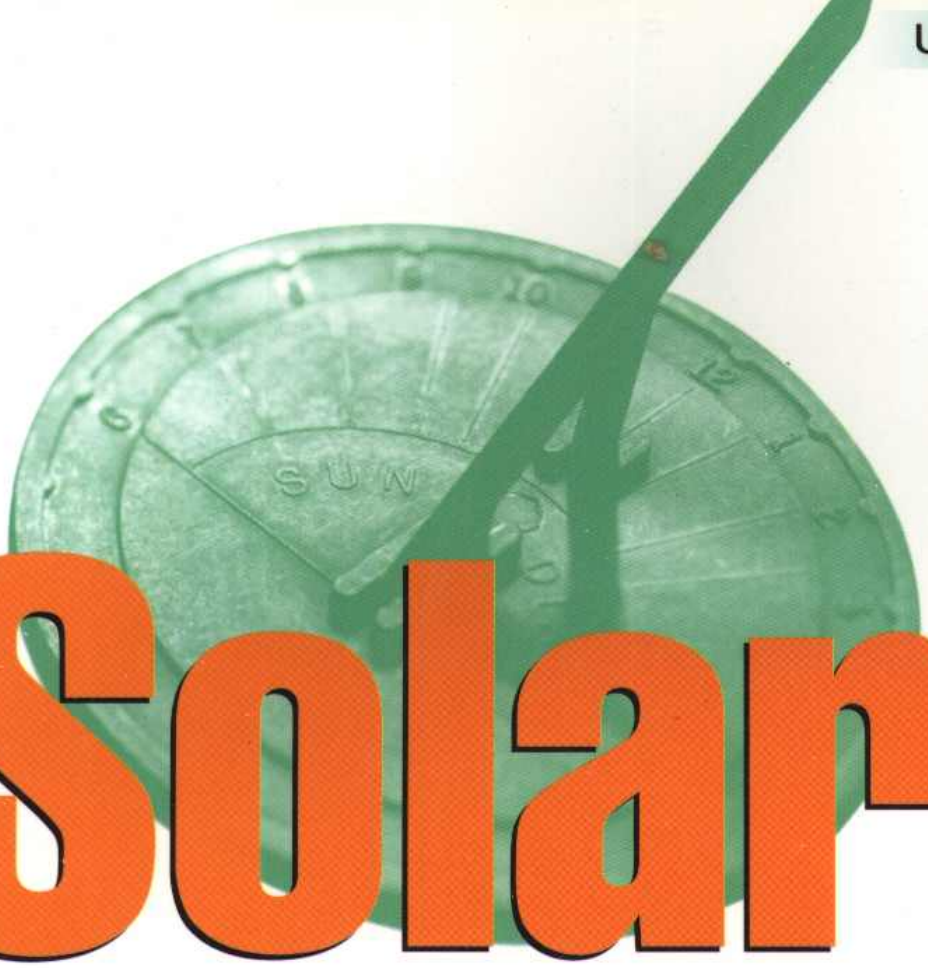
HZ BOOKS

UNIX实用工具译



Solaris

性能管理

Solaris Performance Administration

(美) H. Frank Cervone 著
盛敬刚 徐国平 等译

UNIX 实用工具译丛

Solaris 性能管理

（美）H. Frank Cervone 著

盛敬刚 徐国平 等译

机械工业出版社
China Machine Press

本书特别为Solaris 2.5.1和Solaris 2.6 所写，主要内容包括：性能优化、容量规划、网络配置、内存管理、队列模型、监控活动、内核优化、系统设计等。书中给出了队列模型和性能度量重要的背景理论，以及大量的命令输出和代码示例，拥有从内核结构到网络配置和内存管理的全面的技术和策略信息。无论是新手还是有经验的老手都会从中受益匪浅。

本书适用于UNIX 系统开发人员、网络管理员、系统管理员及相关专业的研究人员。

本书版权登记号：图字：01-2000-1726

图书在版编目（CIP）数据

Solaris 性能管理 /（美）塞奉（Cervone, H. F.）著；盛敬刚等译. -北京：机械工业出版社，2000.11
（UNIX 实用工具译丛）
书名原文：Solaris Performance Administration
ISBN 7-111-08335-0

Ⅰ. S… Ⅱ. ① 塞… ② 盛… Ⅲ. 操作系统（软件），Solaris Ⅳ. TP316.89

中国版本图书馆CIP数据核字（2000）第52748号

机械工业出版社 (北京市西城区百万庄大街22号 邮政编码 100037)
责任编辑：陈贤舜
北京忠信诚胶印厂印刷 · 新华书店北京发行所发行
2000年11月第1版第1次印刷
787mm × 1092mm 1/16 · 13.5印张
印数：0 001-5 000册
定价：25.00元

前　言

一本书的前言有什么作用？首先它是选购指南。好的前言应包括对全书内容的简要概括以及读者需要购买此书的原因，这就是本前言的内容。

本书是对于Solaris环境的性能管理和容量设计的一本综合性指导书。与同类的其他书不同，本书从一个Solaris操作系统的实际使用者的角度出发——而不是从一个Sun公司的员工的角度。因此，我相信这种处理会更中立、更客观，而且比其他书更全面。

本书分为四大部分共18章，第一部分是对于性能分析和容量规划的总体介绍。

第1章是对性能调整和容量规划的简介，探讨了这两方面的内容，包括其中使用的工具和方法。

第2章的重点在于解决如何同时在单处理器系统和多处理器系统上运行Solaris的问题。开始部分讨论了可以使用的不同处理器的类型；其余部分讨论了Solaris的具体实现，包括使用主存储器、高速缓存、虚拟存储器和交换空间。

第3章提出性能的定义，并且描述如何选择度量性能的标准。

第4章重点在于队列模型，这是性能分析的基本组成部分。在描述了队列理论的一些定律后，该章的其余内容考察了进程：离散和连续状态进程的异同、马尔可夫过程的性质以及泊松过程的特性。

第二部分是Solaris的结构综述。

第5章重点是对内核构造的理解。讨论了内核是如何将计算机系统的物理存储器划分为逻辑地址空间的。

第6章集中讨论虚拟存储器以及它如何在机器的物理存储器上实现，尤其是作为一种为支持更多进程而扩展系统容量的机制。

第7章描述Solaris如何实现进程以及如何使用线程来做到这一点。为了理解这些，还研究了轻进程（LWP）、多道处理、抢占、中断、调度和调度类型，以及优先级。

第8章讨论Solaris中文件系统的实现：

- 文件系统的类型
- 文件系统赖以实现的物理介质
- 基于网络的文件系统
- 伪文件系统

虽然这一章的主要部分是关于UNIX文件系统（UFS）的讨论，但同时还讨论了NFS——网络文件系统和DFS——分布式文件系统。

第9章探讨网络和系统间的通信。主要讨论网络的物理实现，既包括本地企业级的也包括全球级的。讨论的问题从基本的输入输出和网络协议开始，直到各种不同的网络拓扑结构，比如以太网、交换以太网、异步传输模式（ATM）、光纤分布式数据接口（FDDI）和令牌环。

第三部分讨论系统性能调节。

第10章研究监控活动的基本过程和命令，以及工作负荷。

第11章是关于存储器管理的内容。这一章说明了存储器问题的主要症状，以及如何解决和避免这些问题。结尾部分讨论了与程序开发、终端用户应用程序执行及系统配置等有关的存储器的注意事项。

第12章集中讨论I/O性能管理，深入研究磁盘性能调整的基础、最大吞吐量的计算、磁盘控制器和接口、廉价磁盘冗余阵列（RAID）的配置以及文件系统的实现和优化。

第13章与网络的物理实现有关。这一章的重点是理解延时、带宽和利用率，这些对于物理性能管理极为重要。明确性能问题和理解网络的使用特点也是这一章讨论的主题。

第14章接着第13章继续讨论网络服务的一些问题，例如NFS、网络信息服务+（NIS+）和域名服务（DNS）。结尾部分给出了关于优化WWW服务器性能的意见和建议。

第15章是这部分的最后一章，主要讨论**/etc/system**文件中的系统参数的调整。

第四部分，也是最后一部分，是关于容量规划工作的内容。

第16章讨论系统活动数据的收集和报告，它们用于收集和解释系统的性能数据。

第17章继续前一章的内容，讨论系统统计子系统。同时使用这些工具以获得系统活动的准确描述。

第18章，即最后一章，讨论容量规划管理，包括容量规划管理的各个主要方面：

- 容量需求的预测。
- 工作负荷的特征。
- 工作负荷模型的开发。
- 期望需求的预测。
- 进一步行动的建议。

本书综合论述了Solaris环境下的性能管理，所以不论是新手还是资深用户都会觉得这本书颇有价值。

不论你是买的还是借的这本书，我都希望你能喜欢它并且在使用Solaris操作系统时有所帮助。

H. Frank Cervone

第一部分 预备知识

第一部分讨论性能调整和容量规划管理的几个基本问题：

什么是性能调整？

什么是容量规划？

什么是系统的主要组件？

在评估和测量性能中的主要因素是什么？

测量性能时用到的理论和公式是什么？

初次接触性能调整或容量规划功能的读者应该阅读这一部分，因为其中定义了许多在本书其余部分用到的术语。有经验的系统管理员可以跳过这一部分，只阅读他们认为有用的部分。

第1章 性能调整和容量规划简介

本章将给出性能调整（performance tuning）和容量规划（capacity planning）的定义，并介绍一些有关的基本概念和术语。

1.1 什么是性能调整

自古以来，人们一直在观察各种活动的进行过程，这些活动包括人类的和机械的。虽然有些人仅仅是出于对观察活动本身的爱好而为之，但是大多数人进行观察的目的是为了预测下一次活动发生的结果。当这些活动是由计算机或计算机系统来进行时，情况也一样。据说在仅出现了两个计算机程序时，第一个面向计算机的性能检查程序就面世了。该计算程序的目的是让计算机操作者知道一个程序在进行计算时到了哪一步。考虑到当时硬件的不稳定性，这些信息非常有价值。当发生了某种错误时，操作者得以知道可以在何处重新开始。

以今天的计算机系统的复杂程度看来，关于性能观察和评估为何如此重要已不再奇怪。性能计算机系统设计和操作的是一个关键标准。显然，人们希望信息处理尽可能快速有效。但是，什么是性能调整？

在本书中，性能调整是指对整个计算机系统的运作进行观察，然后在观察的基础上对系统的不同部件作出调整。最终的结果是整个系统更有效率。这个定义与传统的观点有所不同：我们并不是要使某个特定部件（或者甚至是所有的部件）更有效率，而是要使整系统更有效率。通常这就意味着使所有部件更有效率，但并不一定是如此。可以用一个例子来说明这一点。

考虑这样一种情况：某个系统已经处于不正常状态，对磁盘驱动器的访问缓慢，存储器明显不足。结果是，由于存储器不足，系统使用了相当多的页面调度，而由于磁盘驱动器存在问题，这样做就很慢。那么增加磁盘驱动器的数量或速度对提高性能有用吗？可能有，也可能没有。这是因为在提高系统可承受的页面调度速度时，系统自动地增加了并行任务的数

量，结果导致页面调度速率变得更高，从而使系统再次力不从心。所以，即使有更多的进程在运行，即使磁盘驱动器子系统的性能得到了提高，但由于页面调度工作的增加，在同样的时间内并没有完成更多的工作。结果是系统性能没有得到实际的改善。

以上情形是令人极不希望出现的。在花费大量财力改进磁盘系统后，应该使整体系统性能得到可观的提高。性能调整的最后结果使系统管理员所获得的是：整体系统性能显著地提高。

1.2 什么是容量规划

容量规划是对性能调整的扩充。性能调整用于缓解系统的某些瓶颈或系统速度变慢问题，而容量规划是一种持续的过程，用于确保足够的计算容量以供某个单位使用。所以，根据定义，性能调整是容量规划的一个组成部分。

容量规划的预期结果是这样一种情形：系统的使用者从未经历过降级的服务情况。当然，这是理论上的目标，而且在大多数环境中不可能保证在任何时候任何情况下都能提供足够好的服务质量。但是，这个目标是容量规划者所希望的。如果在大多数时候对大多数使用者都可以保证足够好的服务质量，那么通常可以说容量规划是成功的[⊖]。

容量规划过程的一个基本组成部分是对系统当前状态及其容量的定期报告。报告机制的内容包括系统使用过程中预期出现的变化或更改的细节。如果参与系统使用和操作的各种人员（管理部门，终端用户，系统管理员，系统操作员）对此一无所知，那么容量规划的努力不过是毫无目的的劳动。容量规划的基本目的之一就是在问题发生前对预期的变化作出反应。如果人们对此一无所知，将无法实现这一点。

可以想象，容量规划的最终结果在很多方面和性能调整的最终结果有相似之处。容量规划将给系统管理员一种明确可靠的方法以使系统整体性能保持在稳定的水准。

1.3 工具和方法

性能调整和容量规划涉及很多技术和管理领域（见图1-1）。性能上的问题可能来自许多不同原因。没有一种工具或技术在每一种情况下都能判断出是什么问题。同样，没有一种工具或技术能解决所有问题。

性能调整和容量规划既是一门艺术又是一门科学。当然，在决定系统性能的很多方面，存在标准的步骤和数学公式，但是就像好的艺术一样，并不能机械地得到对系统的成功评估。每一项工程或系统评估都要求了解系统的细节，这需要精心选择工具、方法，对系统进行研究，判断问题的来源。

在性能调整中最重要的工具或许就是观察和直觉。在进行分析之前，必须观察整个计算机系统的现状。在多数情况下，是终端用户将性能问题反映给系统管理员或其他责任部门。虽然从用户那里得到可靠的对问题的全面描述很重要，但是系统性能分析者必须能超出用户的描述范围去发现别的可能引起问题的情况。

举个例子，一名会计用户对系统性能的分析者说，在使用了一个新的结算程序后，系统要花10分钟才能关掉帐簿，而在此之前只用一两分钟。用户由此得出结论：新的结算程序有某种编码错误。但是用户没有告诉也不可能告诉系统程序分析员的是，在使用新的结算程序

⊖ 当然，在某些环境中对可靠性和可用性的要求是百分之百，如航空管理或救生系统。但是，对于大多数商业性和学术性的应用，百分之百的可用性只是理论目标而不是绝对必需的要求。

的几天前，系统使用了一个全新的用户报告系统。当销售部门在运行几个计算量密集的报告的时候，会计部门正要关闭它们。新的结算程序很可能并没有任何错误，问题在于系统负荷比以前大大加重了。尽管应该对结算程序进行研究以发现潜在问题，但系统性能分析员还应该看一看用户报告系统。

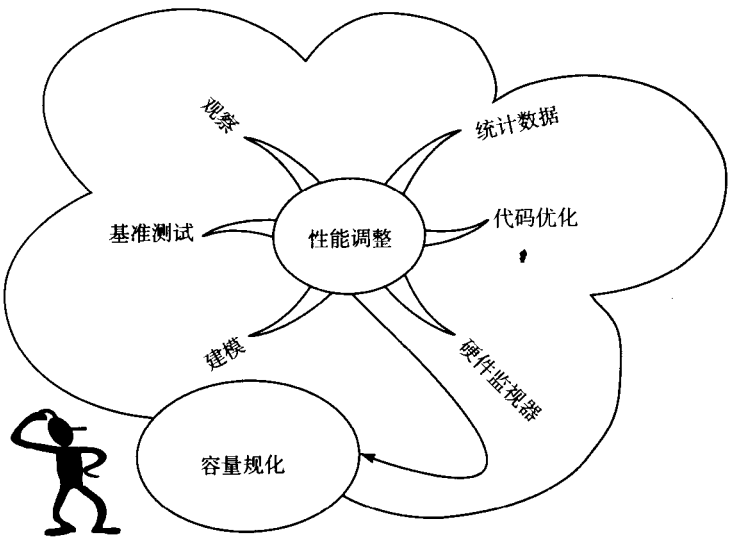


图1-1 容量规划需要多种工具和方法

当然，如果有一个好的容量规划程序，以上情形就也许一开始就不会发生。

统计数据归约程序

在本书中，统计数据是指对计算资源的使用情况采集的信息。在Solaris环境中，不同的程序采集这些信息并报告到/usr/lib/acct子目录中。

统计数据归约程序的起源可追溯到早期计算机程序中插入的检查标记。最后，这些检查标记发展为跟踪例程，当程序运行时，程序员可以记录单步指令的执行。将这些跟踪例程插入到程序的适当位置，就可以收集到一些额外的信息，包括程序运行的情况，以及哪些因素在影响它的运行。但是，这种方法给计算机系统附加了命令，所以也可能对最后的统计数据产生明显的影响。

软件监视器

检查标记和跟踪例程的另一个后继者是软件监视器，在Solaris中是sar程序。软件监视器和统计数据归约程序之间的主要区别在于它们收集不同层次的数据细节。从总体上说，和统计数据归约程序包相比，软件监视器对指令的执行情况检查得更细致。由于软件监视器收集信息的独特途径，这种方法是所有要讨论的方法中最为全面细致的。但是，和统计数据归约程序包一样，这种方法给了计算系统附加的命令，因此可能会明显地影响到采集数据的统计有效性。

程序的优化和程序优化工具

大多数编程语言的编译器都提供选项以优化可执行文件。通常有不同层次上的优化，但

是基本的优化功能都有一些共同之处（见图1-2）：

移去循环内的固定操作；

对共同的子表达式只计算一次；

对寄存器和存储器的使用进行优化；

对数组元素的访问进行优化；

去掉永远不可能执行的代码；

移动子程序代码。

除了编译器可以对代码进行优化外，别的程序也可以在一个程序的执行中观察它的操作过程。这些程序可能是独立的，也可能编译进观察程序中去。它们输出观察程序的执行特性的详细报告。有了这些报告，程序可以把注意力集中在经常使用的那些程序区域中。

未优化代码

```
int process(int factor)
{
  float values[i][j]
  for (j = 0; j<n; j++);                          不正确地按行/列顺序引用数组
   do
     {
      for ( i=0; i<n; i++);
       do
        {
         multiplier = i*factor;                   这步计算可以在该循环外只做一次
         avg(values[i][j], j*multiplier);
         median(values[i][j], j*multiplier);      j* multiplier 表达式应在调用函数前计算一次
         mode(values[i][j], j*multiplier);
        }
  return TRUE;
  printf("Function process complete");            该语句永远不会被执行
}
```

优化后代码

```
int process(int factor)
{
  float values[i][j]
  for ( i= 0; i<n; i++);                          正确地按列/行顺序引用数组
   do
     {
      multiplier = i*factor;                      现在计算是在循环外进行的
      for ( j=0; j<n; j++);
       do
        {
         int j_multiplier = j*multiplier;         j*multiplier 在调用这些函数时只经过一次计算
         avg(values[i][j], j_multiplier);
         median(values[i][j], j_multiplier);
         mode(values[i][j], j_multiplier);
        }
  return TRUE;
}
```

图1-2　代码优化

硬件监视器

硬件监视器是附加在计算机系统内部电路上的电子设备。通过监视系统电路特性的变化，监视器可以检测出系统的使用模式和问题领域。硬件监视器通常只在特殊领域中使用，最常

见的是用于监视网络传输情况。作为一种通用的计算系统监视工具，它对于大多数性能分析者并不一定有用。这是因为使用硬件监视器需要预先进行正规的训练和实践，包括监视器的使用方法、被监视的计算系统的详细结构以及对计算系统工作负荷的详尽了解。除此之外，硬件监视器不论是购买（或租赁）还是使用都非常昂贵。

基准测试

基准测试通常用于建立一个标准，用于进行不同计算系统之间或不同设置的同类系统之间的比较。

典型的基准测试方法是用几个程序（或几套程序）来代表程序用户“标准的”或“典型的”工作负荷。

与此同时，基准测试与前面所说的工具还有一点不同：它是作为一种预测性的方式使用而不是作为测量工具。前面讨论的工具仅仅测量现有工作负荷，它们没有也不能预测未来的使用情况。但是，基准测试最常用来预测在特定的设置下给定一套环境参数时会发生的情况。

一套标准的基准测试程序在容量规划过程中是非常有价值的工具。通过在适当的时机运行这些是基准测试，性能分析者可以得知系统设置的持续有效性。通过跟踪这些基准测试的运行情况，性能分析者可以更好地预测系统需要在何时进行更改或升级。

建模方法

建模方法使用模拟来预测一个计算系统的性能。通过建立目标系统的数学模型，就有可能预测一个特定的动作对于一个研究的或创建的系统将产生什么影响。在某些情况下，对目标系统本身直接进行监视太昂贵、太费时或太危险，就往往使用建模方法。

建模方法的优势在于它可以在创建或购买一个系统之前对系统的性能进行检查。所以，前面讨论的方法是为了解决问题，而建模方法可以使问题在发生前就得到解决。

工具和方法小结

总的来说，在性能监测中最常用的是统计数据归约程序包、软件监视器和程序优化。当对网络进行分析时，还应包括硬件监视器。在容量规划中，基准测试和建模方法用于使用性能监视中采集的信息和建立长期的规划。

1.4 容量管理

确保一个组织拥有足够计算容量的任务称作容量管理，这个过程包括性能调整和容量规划（见图1-3）。虽然性能调整是容量规划的必要前提，但是性能调整也有其自己的准则和功能。性能调整主要关心短期结果和利益。容量规划是预测性的，并且主要关心长期结果和利益。所以，容量管理是这些因素的结合。特别地，容量管理方案保证该组织为现有用户提供可接受的服务，

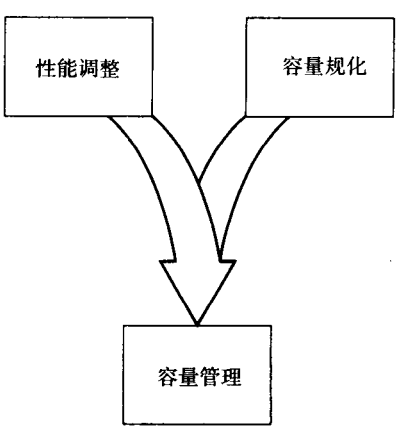


图1-3 容量管理是性能调整和容量规划的结合

同时还要为将来的工作负荷提供足够容量。

成功的容量管理程序由几个有内在联系的部分组成（见图1-4）：

管理协定（对人员和设备）；

对容量规划必要性的认识；

适当工具的使用；

对用户的影响；

熟练的人员。

对于一个组织的计算需求的满意程度来说，容量管理是一个极其重要的因素。如果上述组成部分中有一个不在容量管理项目中，该项目最后都将失败。最终这将给该组织的整个计算环境带来有害的影响，所以对于任何组织的长期利益来说，有一个容量管理项目并且全力支持该项目是必不可少的。

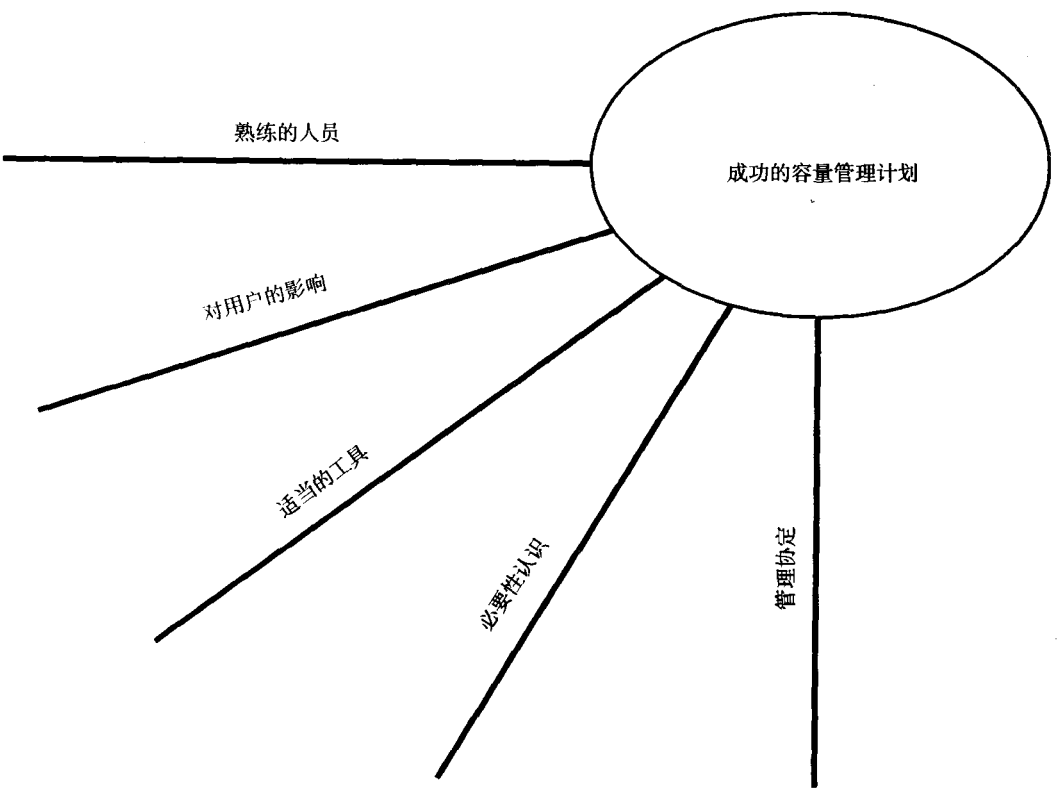


图1-4　成功的容量管理包含不同级别的综合因素

容量管理的阶段循环

容量管理是一个循环过程，它永不停止。不管一个系统今天调整得多好，明天总会有新的变化打破平衡。这个过程总是在进行中，并且根据系统的复杂性，在不同时刻处于不同的阶段，认识到这一点很重要。但是，容量管理循环的主要阶段是什么呢？尽管可能有不同的叙述，这里将主要讨论四个阶段（见图1-5）：

现有系统分析；

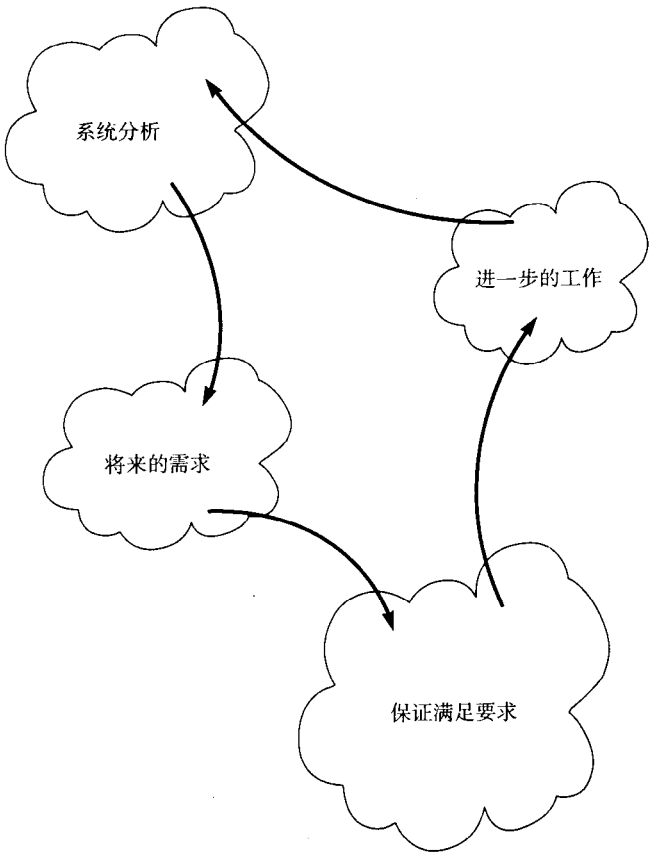


图1-5 容量管理生命周期

决定将来的需求；

确保满足需求；

进一步的工作。

现有系统分析。在这一阶段中，将决定容量管理计划的基本目标。挑选有关负责团体，并且如果必要的话，就他们所需要掌握的知识进行训练。

在实际系统中，现有的工作负荷已是经过分析的。如果还没有，那么文档记录了系统的配置，包括硬件和软件。这个阶段将对用户进行服务级需求的咨询。典型的服务级协定包括：

- 对可获得服务的需求，比如每星期7天，每天24小时；
- 事务响应时间，比如在点击确定后，10秒内得到所需的数据；
- 问题处理程序，比如从星期一至星期六上午6点至午夜12点，有人随时准备处理系统瘫痪问题。

决定将来的需求。在这个阶段中，性能分析者需要评估该组织对未来计算需求的计划。这个过程称作工作负荷预报。分析者将根据该组织各个部门的系统计划，试图预测出未来的计算工作负荷。这些计划通常以“用户方式”交给分析者，所以必须将其转译成可测量的标准。例如，分析者可能会对将来的客户订单做个估计，并根据预期的事务复杂性将它们分成不同的处理类别。这样，分析者就可以推断出预期的容量；这将和代价分析一起提交给管理部门。

在这个阶段中，性能分析者还对实际工作负荷和先前的预计值进行比较。若有不符之处，则应当研究是什么因素导致预测不准。

确保满足需求。在这一阶段，分析者监督容量扩充设备的安装，并衡量安装后容量的增加。虽然安装容量扩充设备可能是另一个人或部门的事情，但是性能分析者有必要确认安装就绪。

在这一阶段中，测量容量的增加要求很苛刻。确保的确获得了预期的增加是极重要的。更进一步讲，分析者必须复核哪些在容量扩充过程中使用的方法是合理有效的。如果未能达到预期的目标，有必要回头检查为什么偏离预期结果。正是在这一阶段中，基准测试在容量扩充前后的运行，是最有益和具有启示性的。

进一步的工作。性能分析者必须随时与应用开发部门保护联系。从一开始就设计出好的性能比在程序完成后进行修改要容易得多。

有必要将过去的工作负荷和服务信息不断地收集起来，并整理成有用的数据。另外，系统配置记录和工作负荷预报也要保存以备查阅。对于系统容量的分析和控制来说，所有这些信息来源都是必要的。

此外，性能分析者还必须不断了解新的技术发展情况，以及它们将对以后的规划产生什么样的影响。

其他因素

一个性能分析者的工作中有很大一部分不是技术性的而是属于“政治性”的。性能分析者必须能够周旋于用户和管理部门之间，能够说服别人接受自己的观点。性能分析者必须乐于听取外部的意见和建议。拒绝对一个新主意进行尝试可能会毁掉整个容量规划进程。

最后，性能分析者的职能必须独立于该组织计算机构的其他职能。例如，如果容量规划工作受应用开发部门领导，那么将会对应用项目的调整作出太多的努力，以至损害了其他性能调整。对操作或系统管理负责也会导致类似的结果。在不可能做到职能独立的情况下，容量管理职能最好被安排为向系统管理部门或相当的具体部门负责。这样做的部分原因是基于传统的信息系统部门结构，同时也是因为系统管理部门对整个计算机环境的涉及通常最为广泛。

1.5　小结

本章是对性能调整和容量规划的简介。我们了解了性能调整和容量规划的意义，并且探讨了其中使用的工具和方法。这些工具和方法包括：

- 统计数据归约程序；
- 软件监视器；
- 程序优化和程序优化工具；
- 硬件监视器；
- 基准程序；
- 建模方法。

最后，以关于性能调整和容量规划如何组成容量管理的讨论作为本章的结束。

在下一章中，将回顾计算机系统的各个组成部分。

第2章 计算机组成综述

本章的目的在于复习Solaris系统有关的软件、硬件构成的一些概念和词汇。

通常分析和诊断性能问题时，会把硬件分为四个主要部分（见图2-1）：

- 处理器（中央处理单元）系统（寄存器，高速缓存）；
- 存储器系统（主存，高速缓存，I/O缓冲器，网络缓冲器）；
- I/O系统；
- 网络系统。

后面的讨论就是按这种系统划分进行的。

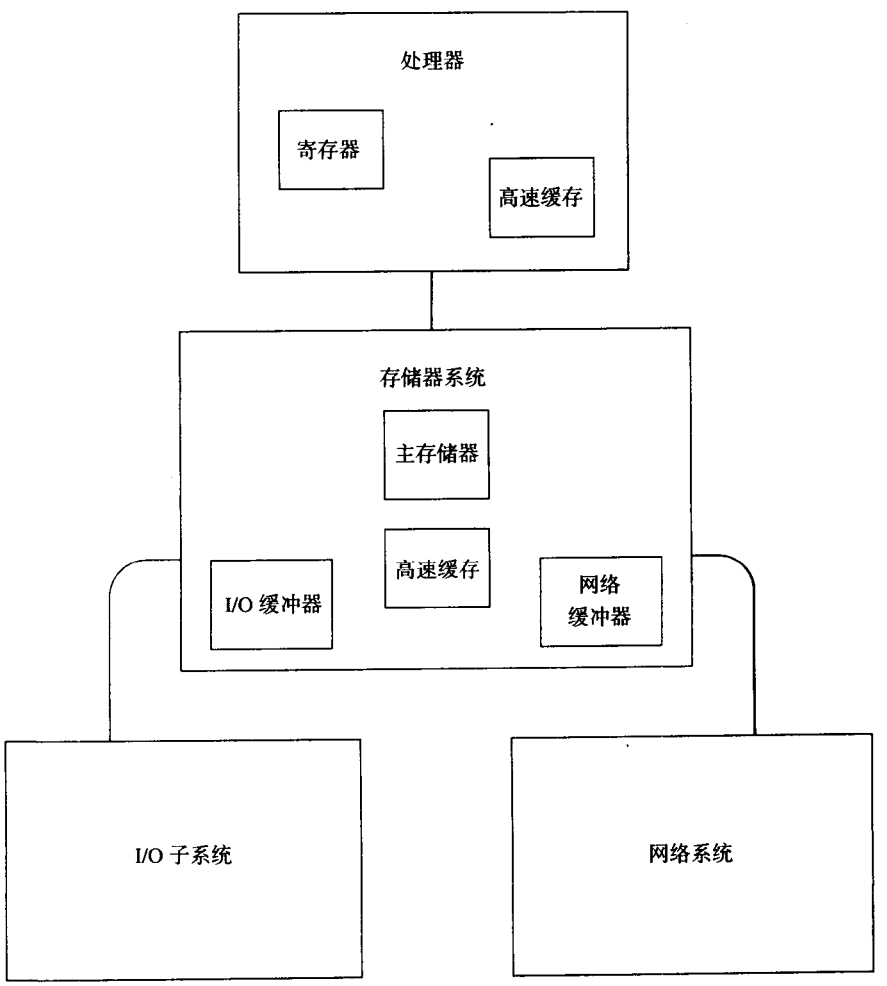


图2-1 存储层次图

性能问题并不仅仅由硬件决定。操作系统（或与之相关的模块）和应用程序对系统的整

体性能有着重要影响。很多情况下，系统性能没有达到最佳状态不是因为硬件功能的问题，而是由于操作系统或应用程序的配置不当造成的。有时，附加硬件以期解决上述问题确为有效，至少是一段时间有效。而在其他许多情况下，这样做并不能缓解由软件产生的问题。为了使性能达到满意的程度就必须找出软件中的问题并加以解决。

2.1 处理器系统

处理器最主要的功能就是执行“处理”数据的指令。在大量的商业用途和教育领域，计算机系统的主要处理功能可能是关于数据格式的转换或数据的传输。而在科学和工程上，最主要的则是数据运算。

这两种情况下工作负荷的特性差别很大。前者的重点是在于输入和显示大量的各种数据；后者的重点则在于处理规模较小但更为关联的数据。因此，前者的性能问题可能更多地是关于I/O功能的；而对于后者，系统的处理能力则可能是最主要的问题。由于不同计算机系统功能要求差别很大，应具体情况具体分析。

单处理器系统

单处理器系统是只有一个处理器单元的计算机，工作站和中小型服务器通常属于这一类，其中也包括大部分的Solaris系统。

在结构上，单处理器系统比多处理器系统简单，多处理器系统指含有两个或两个以上处理器的系统。这里的简单既指硬件也指软件[⊖]。硬件上，部件和互联较为简单；在软件上，操作系统也少做一些事情，比如：怎样保持各CPU同步存取存储器，协调程序在不同处理器上的执行。

处理器本身由硅芯片构成，被封在方形陶瓷片中。电气引线从底部引出，排列成特定方式，以便插入主系统板上的处理器插座上。处理器正确插入插座后，就与主板连接好了。通过主板，处理器可以连通系统总线。总线用于系统和I/O子系统的各部件间的数据传输。

不同处理器之间的区别有以下几项：

体系结构；

处理器内部同时操作的数据长度；

数据通道宽度；

寻址能力；

时钟频率或执行指令的速度。

在处理器中指令的执行可以分为几个阶段（或部分）[⊜]。预取单元从存储器得到指令，然后把指令放入指令队列（或管道）。解释单元（或解码）把队列中的指令翻译成控制单元能够接受的格式。控制单元监督指令的执行，主要是确认存储器的存取地址是合法的以避免发生存储器冲突。如果指令是浮点指令，将被传到浮点运算单元（FPU）执行，否则送到算术逻辑单元（ALU）执行。ALU负责处理所有逻辑操作和无需FPU处理的数学运算。调页和调段单元负责把程序中有关的逻辑地址译成存储器的物理地址。

另外还有两个单元：总线接口单元和高速缓存。总线接口单元管理处理器与系统总线间

⊖ 多处理器系统的内存复杂性在下一节讨论。

⊜ 这指的是基于Intel 的处理器。然而，基本的处理器与SPARC 处理器相同。

的数据和指令的传输；高速缓存用于预取单元从主存预取指令。由于cache比主存快得多，并且cache控制器试图预测下一个要用的数据和指令，因此指令从cache送到指令队列比从主存到指令队列要快得多。

这里有一个最困难的问题，就是预测性能在给定指令序列情况下的执行时间是变化的。特别是在单CPU系统中这一点尤为突出，这是因为单处理器要响应管理系统里发生的每件事情。I/O通常不被认为会消耗大量CPU资源，但如果单处理系统有大量I/O请求将给处理器带来非常大的负担，这是因为处理器将忙于响应I/O设备产生的频繁的中断信号。

多处理器系统

多处理器系统有两个以上的处理器。在紧耦合环境下，各处理器一起工作，由单一的操作系统控制，共享存储器和系统的其他设备。所有Solaris系统均运行在紧耦合模式下。在松耦合环境下，处理器间没有共享存储器或其他设备。

除了上述分类，按操作系统也可分为对称的与非对称的。在非对称系统中，一个处理器（称为主处理器）专门用于运行操作系统内核程序。这个处理器处理所有的内核功能：I/O、调度、虚拟内存操作和系统管理。其他的处理器只用于运行用户程序。这种安排的问题是，当有大量的系统请求时，性能将大大下降，因为这些请求不能分配给多个处理器，运行用户程序的处理器可能长时间等待主处理器。

在对称系统中，操作系统内核被分为几个部分，如Solaris 2中的线程（thread）。每个线程是独立的功能，可在任何一个处理器上运行。因此，向内核的请求可以分配到任一个可用的处理器，这减少了用户进程等待内核功能完成的时间。Solaris 2被称为高度对称的操作系统不是指所有的内核功能都对称，而是大部分功能可以实现对称。

多处理器系统并没有因为增加新的处理器而使性能线性地提高，因为要为管理增加的处理器加大开销。例如：不可能简单地增加一个处理器就使系统性能提高一倍。事实上，多处理器系统增加太多处理器可能会降低系统的性能，这是因为用于管理多处理器的开销超过了增加的处理器的处理能力。

协处理器

在基于Intel处理器的机器上，某些处理器没有内置浮点单元。这些处理器可以通过增加一个处理器浮点运算的协处理器（coprocessor）来提高性能。没有协处理器，主处理器必须进行仿真来处理所有浮点指令，这会严重降低总的处理速度。当采用了协处理器后，结合主处理器，浮点指令可以同主处理器并行执行。

只有Intel的386SX和486SX需要外部协处理器，386DX和486DX、Pentium（奔腾）、Pentium Pro和Pentium Ⅱ处理器已经把浮点处理器与主处理器整合为一个芯片了。

2.2 存储器系统

存储器系统可分为4个类：

外存：如磁盘和磁带；

交换空间或虚拟存储器；

主存；

高速缓存。

本节将讨论后面三项，因为第一项作为I/O系统一部分来讨论。

主存储器

所有计算机都有主存，无论是怎么制造的、什么模式或什么操作系统。这种存储器也称为实存储器或核心存储器（内存）。当处理器执行程序中的指令时，先从主存中读取指令和数据。它们都是在处理器读取之前就已经装入主存了。

但主存还不如处理器那么快。因此处理器可能要不时地等待主存的程序指令和数据。为了减少这个期间，大多数系统采用了高速缓存来作为中介以加速处理器到主存的存取。

高速缓冲存储器

高速缓冲存储器（cache）用于从主存中预取指令和数据。在主处理器执行程序时，一个附加的逻辑单元将预测处理器下几步需要的指令是什么，并根据预测结果把指令和数据从主存中读到高速缓存中。如果预测正确，主处理器直接从高速缓存读取数据和指令，将比从主存读取节省3~4倍的时间。这是因为主存的典型存取时间为60~70 ns而高速缓存通常为15~20 ns。

通常高速缓存比主存贵得多，cache与主存的连接不是对应的。另一个情况是系统上每个处理器的cache数量有一个最佳值，多出的cache不会再提高处理器的速度。

很多新型处理器中，cache又被分为两级（见图2-2）：处理器内置cache 作为处理器与主板上的cache 的缓冲。

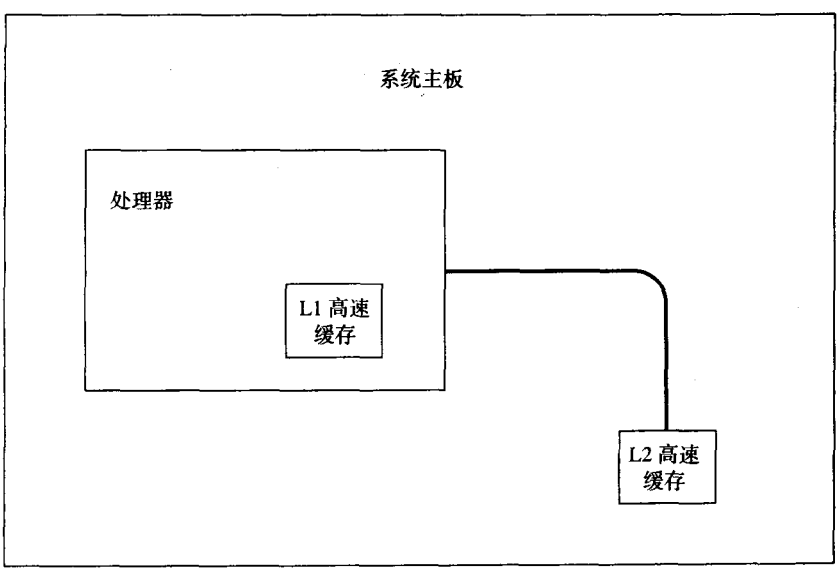


图2-2　高速缓冲存储器的交互作用

虚拟存储和交换区

虚拟存储或交换区[⊖]使得系统能够处理所需存储超过物理内存的程序。例如：虚存技术可使

⊖ 在Solaris 环境中，尽管这两个术语是不同的，却经常互用。虚拟存储在磁盘上指定为交换区的部分实现。