SYBASE®
An SAP® Company

性能和调优系列：
基础知识

Adaptive Server® Enterprise

15.7

文档 ID：DC01071-01-1570-01

最后修订日期：2011 年 9 月

Sybase, Inc., One Sybase Drive, Dublin, CA 94568。

目录

第 1 章

基础知识简介

优良性能

性能是衡量一个应用程序或在同一环境下运行的多个应用程序的效率的尺度。性能通常使用**响应时间**和**吞吐量**来衡量。

响应时间

响应时间是完成单项任务所需的毫秒数、秒数、分钟数、小时数或天数。缩短响应时间的方法有：

- 通过查询调优和索引使查询、事务和批处理更高效
- 使用更快的组件 （例如，更快的客户端和服务器处理器以及更快的磁盘和存储）。
- 使等待时间最短 （例如，通过改善网络、物理和逻辑锁争用）

在某些情况下， Adaptive Server® 已自动优化为缩短初始响应时间，即，将第一行返回给用户所需的时间。当用户使用查询检索多行然后使用前端工具缓慢浏览它们时，这尤其有用。

吞吐量

吞吐量指每个时间单位完成的工作量。例如，执行的工作量采用：

- 单个事务的数量（例如，每秒 100 个从 Wall Street 插入交易的事务）。
- 整个服务器上的所有事务 （例如，每秒 10,000 个 read 事务和每秒 1,500 个 write 事务）。
- 执行的 read 数 （例如，每小时的特定查询或报告数）。

不过，在调优 Adaptive Server 以缩短响应时间时，可能会降低吞吐量，反之亦然。例如：

- 添加索引可为使用这些索引来避免更昂贵的扫描的查询以及 update 和 delete 提高性能。不过，必须在数据操作语言 (DML) 操作期间维护索引，这可能降低性能。
- 使用异步磁盘写入可缩短包括单个用户的单个事务的响应时间，但异步磁盘写入会降低多用户吞吐量。

性能设计

大多数性能提高源自良好的数据库设计、透彻的查询分析以及妥当的索引编排。在开发应用程序时，通过构造优良的数据库设计和使用 Adaptive Server 查询优化程序，可最大限度地提高性能。

还可以通过分析应用程序如何与 Adaptive Server 一起工作来提高性能。例如，客户端可能最初将大小为 1KB 的行发送给 Adaptive Server，然后等待 Adaptive Server 确认收到这些行，才发送下一行。您可能发现如果客户端合并或批处理它发送给 Adaptive Server 的行，从而大大简化该过程并且减少 Adaptive Server 和客户端之间的交互需求，客户端和 Adaptive Server 之间的性能将提高。

还可以使用硬件和网络分析来查找安装中的性能瓶颈。

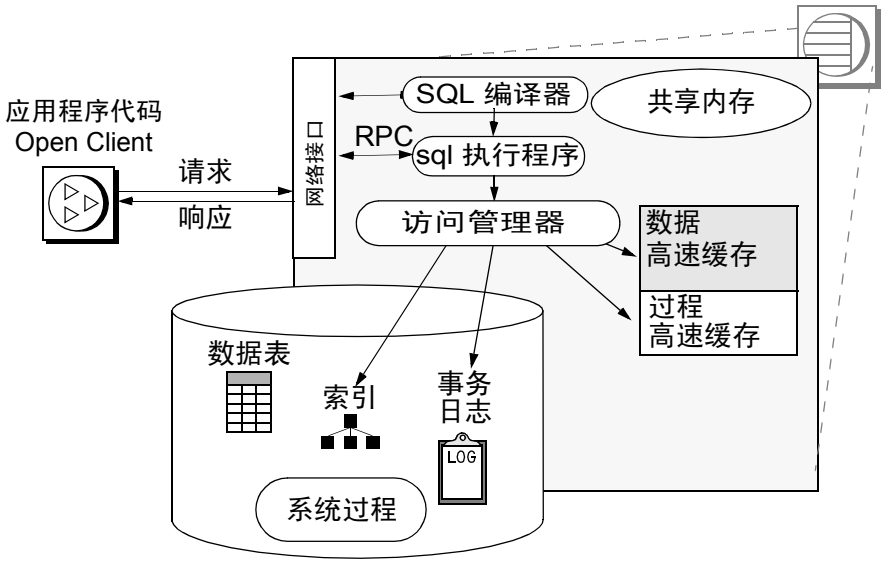
调优性能

调优可提高性能并减少争用和资源消耗。系统管理员查看：

- 系统调优 — 调优整个系统。请参见 《性能和调优系列：物理数据库调优》。
- 查询调优 — 使查询和事务更快，并使全局和物理数据库设计更高效。请参见 《性能和调优系列：查询处理和抽象计划》。

使用 Adaptive Server 及其环境的此系统模型来确定每一层上的性能问题。

图 1-1：Adaptive Server 系统模型



系统调优的主要部分是减少对系统资源的争用。随着用户数量的增加，对资源 （如数据和过程高速缓存、系统资源的螺旋锁及 CPU）的争用会增加。逻辑锁争用的可能性也随之增加。

调优级别

Adaptive Server 及其环境和应用程序可细分为组件 （或调优层），以便在分析时将系统组件隔离出来。在许多情况下，必须对两层或更多层进行调优，以使它们实现最佳合作。

在某些情况下，消除某一层上的资源瓶颈可暴露出另一问题区域。乐观地说，解决一个问题有时可减轻其它问题。例如，如果用于查询的物理 I/O 速率较高，您因此添加了更多的内存来缩短响应时间和提高高速缓存命中率，就可能减轻磁盘争用所引起的问题。

应用层

大部分性能增益来自查询调优（前提是数据库设计优良）。在应用层上，以下问题是相关的：

- 决策支持系统 (DSS) 和联机事务处理 (OLTP) 需要不同的性能策略。
- 事务设计可能会降低性能，因为长时间运行的事务会持有锁并减少对数据的访问。
- 修改数据需要相关的表连接，以满足关系完整性的要求。
- 支持选择的索引会增加修改数据所需时间。
- 安全性审计可能会限制性能。

解决这些问题的可选方案有：

- 用于将决策支持移出 OLTP 计算机的远程或复制处理
- 可减少编译时间和网络使用的存储过程
- 可满足应用程序需求的最低锁定级别

数据库层

应用程序共享数据库层的资源，包括磁盘、事务日志和数据高速缓存。

一个数据库可有 2^{31} (2,147,483,648) 个逻辑页。这些逻辑页分布在不同的设备上，最多可达每个设备的上限。因此，最大数据库大小取决于可用设备的数量和大小。

“开销”是为服务器保留的空间，不可用于任何用户数据库。通过合计以下各项来计算开销：

- master 数据库的大小，加上
- model 数据库的大小，加上

- tempdb 的大小，加上
- （对于 Adaptive Server 版本 12.0 及更高版本） sybsystemdb 的大小，加上
- 服务器的配置区域使用的 8KB。

在数据库层上，影响开销的问题包括：

- 开发备份和恢复方案
- 跨设备分配数据
- 运行审计
- 可降低性能并将用户锁在表外的高效调度的维护活动

通过以下方法解决这些问题：

- 使用事务日志阈值自动进行日志转储以避免空间问题
- 使用阈值监控数据段中的空间
- 添加分区来加快数据装载和查询执行
- 将对象分配给各设备，以避免磁盘争用或利用 I/O 并行度
- 采用高速缓存以提高关键表和索引的可用性

Adaptive Server 层

在服务器层上有许多共享资源，包括数据和过程高速缓存、线程池、锁和 CPU。

Adaptive Server 层上的问题包括：

- 支持的应用程序类型：OLTP、 DSS 或混合。
- 支持的用户数量 — 随着用户数量的增加，对资源的争用会发生变化。
- 线程池中的线程数。
- 网络负担。
- 当用户数量和事务处理速率达到较高级别时， Replication Server® 或其它分布式处理会出现问题。

通过以下方法解决这些问题：

- 对内存 （最关键的配置参数）和其它参数调优
- 决定一些处理可在客户端进行
- 配置高速缓存大小和 I/O 大小

- 重新组织线程池
- 添加多个 CPU
- 调度批处理作业并报告非工作时间
- 为转变工作负荷模式重新配置某些参数
- 确定是否将 DSS 移动到另一个 Adaptive Server

设备层

设备层与存储数据的磁盘和控制器相关。 Adaptive Server 可管理几乎无限数量的设备。

设备层的问题有：

- 系统数据库、用户数据库和数据库日志在各设备上的分布
- 是否需要为提高堆表的并行查询性能或插入性能使用分区

通过以下方法解决这些问题：

- 使用多个中等大小的设备和控制器；这样可提供比使用少量大型设备更好的 I/O 吞吐量
- 分配数据库、表和索引，使所有设备间 I/O 负载均匀
- 使用段和分区以提高在并行查询中使用的大型表的 I/O 性能

注释 Adaptive Server 设备映射到操作系统文件或裸分区，并依赖基础物理设备和磁盘的性能。可能在操作系统级别上发生争用：控制器可能饱和，并且以存储区域网络逻辑单元号 (SAN LUN) 组织的磁盘可能过度工作。在分析性能时，平衡物理负载以在操作系统设备、控制器、存储实体和逻辑单元号上正确分配负载。

网络层

网络层与将用户连接到 Adaptive Server 的一个或多个网络相关。

几乎所有 Adaptive Server 用户都可通过网络访问其数据。

此层上的问题包括：

- 网络通信量
- 网络瓶颈
- 网络速度

通过以下方法解决这些问题：

- 配置包大小以与应用程序需求相适应
- 配置子网
- 隔离重网络负荷用户
- 移到高容量网络
- 设计应用程序，限制其所需要的网络通信量

硬件层

硬件层涉及 CPU 和可用内存。

硬件层的问题有：

- CPU 吞吐量
- 磁盘访问：控制器及磁盘
- 磁盘备份
- 内存使用
- 虚拟机配置：资源分配

通过以下方法解决这些问题：

- 添加与工作负荷相匹配的 CPU
- 配置管家任务以提高 CPU 利用率
- 遵循多处理器应用程序设计准则以减少争用
- 配置多个数据高速缓存

操作系统层

理想情况下， Adaptive Server 是计算机上唯一的主要应用程序，并且只能与操作系统和其它 Sybase® 软件（例如 Backup Server™）共享 CPU、内存和其它资源。

操作系统层上的问题包括：

- 可用于 Adaptive Server 的文件系统
- 内存管理 — 精确估算操作系统开销和其它程序内存使用率
- CPU 可用性和分配到 Adaptive Server

通过以下方法解决这些问题：

- 考虑网络接口
- 在文件和原始分区之间选择
- 增加内存大小
- 将客户端操作和批处理作业转移到其它计算机
- 对 Adaptive Server 使用多个 CPU

确定系统限制

CPU、磁盘子系统和网络的物理限制施加性能限制。延迟或带宽限制由设备驱动程序、控制器、交换机等施加。可以通过添加内存、使用更快的磁盘驱动器、切换到带宽更高的网络和添加 CPU 来克服其中一些限制。

线程、线程池、引擎和 CPU

在进程模式中，Adaptive Server 通常对每个已配置的引擎使用一个 CPU。在线程化模式中， Adaptive Server 通常对每个已配置的引擎线程使用一个 CPU，并且对非引擎线程使用附加 CPU，例如 syb_system_pool 中的 I/O 处理线程。

不过，CPU 的定义在现代系统中不明确。操作系统报告为 CPU 的对象可能是多核处理器或子核线程的核心，其中每个核心支持多个线程（通常称为超线程、SMT 或 CMT）。例如，具有两个 4 核处理器（每个核心 2 个线程）的系统报告它具有 16 个 CPU。这不意味着所有 16 个 CPU 都可用于 Adaptive Server。

Sybase 建议您确定可用于 Adaptive Server 的实际 CPU 数量以及每个引擎和每个非引擎线程需要的 CPU 处理能力。良好估计是允许操作系统使用一个 CPU。在线程化模式中，还允许 I/O 线程使用一个 CPU。

基于此建议，在进程模式中，在包含 16 个 CPU 的系统上配置的引擎数不超过 15 个，而在线程化模式中，在同一系统上配置的引擎数不超过 14 个（在线程化模式中，每个引擎可以执行比在进程模式中更有用的工作）。

配置 CPU 时，请考虑：

- 具有高 I/O 负载的服务器可能需要为 I/O 线程保留更多 CPU。
- 并非所有 CPU 都相同，您可能无法保留所有子核线程。例如，您可能需要将 8 核、 16 线程系统视为仅有 12 个 CPU。
- 可能需要为主机上的其它应用程序保留 CPU。

Sybase 建议您使用 sp_sysmon 验证配置。如果出现大量非自愿环境切换，或引擎时钟周期利用率高于操作系统线程利用率，则可能相对于基础 CPU 过度配置了 Adaptive Server，这可导致吞吐量显著降低。

各种逻辑页大小

dataserver 二进制文件建立主设备 （位于 *$SYBASE/ASE-15_0/bin* 中）。dataserver 命令可用于创建逻辑页大小为 2KB、4KB、8KB 或 16KB 的主设备和数据库。较大的逻辑页可为一些应用程序提供好处：

- 可以创建比使用较小页大小时更长的列和行，从而允许表更宽。
- 根据应用程序的性质，可以提高性能，因为 Adaptive Server 每次读取页时可以访问更多数据。例如，读取单个 16K 页时引入高速缓存中的数据量是读取 2K 页时的 8 倍；读取 8K 页时引入的数据是读取 4K 页时的 2 倍，依此类推。

  不过，在使用较大页时，仅访问表中的一行的查询 （称为点查询）使用占据较多内存的行。例如，保存到配置为 2k 逻辑页的服务器的内存中的每行使用 2k，但如果服务器配置为 8k 逻辑页，则保存到内存中的每行使用 8k。

  分析单个案例来检验使用大于 2KB 的逻辑页是否有益。

列数目和列大小

表中可创建的列的最大数量为：

- 所有页锁定表和 DOL 锁定表中的固定长度列 — 1024。
- 所有页锁定表中的可变长度列 — 254。
- DOL 锁定表中的可变长度列 — 1024。

列的最大大小取决于：

- 表是否包含可变长度列或固定长度列。
- 数据库的逻辑页大小。例如，在具有 2K 逻辑页的数据库中，所有页锁定表中的最大列大小可与单个行的大小相等，约为 1962 个字节，但要减去行格式的开销。同样，对于 4K 页，所有页锁定表中的最大列大小可达 4010 个字节，但要减去行格式的开销。

表达式、变量和存储过程参数的最大长度

传递给存储过程的表达式、变量和参数的最大大小为 16384 (16K) 字节，对字符或二进制数据的任何大小的页都一样。无须使用 writetext 命令就可以将最大大小的变量、文字插入到文本列。

登录数

表 1-1 列出对 Adaptive Server 的登录数目、用户数目和组数目的限制。

表 1-1：登录数目、用户数目和组数目的限制

项目	版本 15.0 限制
每台服务器的登录数 (SUID)	2147516416
每个数据库的用户数	2146585223
每个数据库的组数	1032193

限制对性能的影响

对 Adaptive Server 设置的限制指的是服务器可能必须为单个查询、DML 操作或命令处理大量数据。例如，如果使用具有 char(2000) 列的 DOL 锁定表， Adaptive Server 必须分配内存才能在扫描表时执行列复制。在运行查询或命令的过程中，增加内存请求意味着可能要减少吞吐量。

内核资源内存的大小

内核资源内存是高速缓存。 Adaptive Server 将内核资源内存保留为池，所有线程池从其中接收内存。可分配给内核资源内存的最大大小为 2147483647 个 2K 逻辑页。

分析性能

发生性能问题时，请确定问题的起因及解决问题时要达到的目标。

分析性能问题：

1　收集性能数据以获得基本测量值。例如，可以使用以下一种或多种工具：

- 内部开发的基准测试或行业标准的第三方测试。
- sp_sysmon，一个用来监控 Adaptive Server 性能并提供描述 Adaptive Server 系统行为的统计输出的系统过程。

  请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。
- 监控表，它们从服务器范围及用户或对象级别角度说明资源利用率和争用。
- 任何其它适用工具。

2　分析数据以了解系统和所有性能问题。编制问题清单并回答问题，以对 Adaptive Server 环境进行分析。该清单可能包括如下问题：

- 问题有哪些症状？
- 系统模型的哪些组件对问题有影响？
- 该问题影响所有用户还是只影响某些应用程序的用户？
- 该问题是间歇性的还是持续性的？

3　确定系统要求和性能目标：

- 此查询执行的频率是多少？
- 要求的响应时间多长？

4　确定 Adaptive Server 环境 — 了解所有层的配置和限制。

5　分析应用程序设计 — 检查表、索引和事务。

6　根据性能数据，提出有关性能问题的可能原因及可能解决方案的假设。

7　最后，选择执行下面的解决方案以检验该假设：

- 调整配置参数。
- 重新设计表。
- 添加或重新分配内存资源。

8 选用步骤 1 中用于收集基本数据的测试工具来确定调优效果。性能调优常常是一个重复的过程。

如果基于步骤 7 采用的操作不能满足在步骤 3 中设定的性能要求和目标，或者在一个区域进行的调整引起了新的性能问题，则从步骤 2 起重复此分析。可能需要重新评估系统要求和性能目标。

9 如果测试表明假设是正确的，则在开发环境中执行该解决方案。

www.sybase.com 包括讨论其它性能分析方法的白皮书。

范式

规范化是关系数据库设计过程的组成部分，可用于重新组织数据库，以便尽可能减少和避免不一致和冗余。

不同的范式组织管理员信息，以促进高效维护、存储和数据修改。规范化简化了查询和更新管理，包括数据库的安全性和完整性。不过，规范化通常会创建更多数量的表，而这又增加了数据库的大小。

数据库设计人员和管理员必须决定最适合其环境的各种方法。

锁定

Adaptive Server 锁定活动事务当前使用的表、数据页或数据行来保护它们。多用户环境需要锁定功能，因为几个用户可能同时使用同一数据。

当一个进程持有阻止另一进程访问数据的锁时，锁定会影响性能。被锁阻止的进程将休眠，直到锁被释放。这种情况称为锁争用。

死锁是会给性能带来更严重影响的锁定。当两个用户进程各自持有对不同的页或表的锁，而每个进程又想获取对方进程所持有的页或表的锁时，就会发生死锁。累计占用 CPU 时间最少的事务将被注销，它的所有工作都将被回退。

了解 Adaptive Server 中锁的类型有助于减少锁争用，避免或尽可能减少死锁。

Performance and Tuning Series: Locking and Concurrency Control （《性能和调优系列：锁定和并发控制》）讨论锁定的性能影响。

特殊注意事项

在 Adaptive Server 中创建数据库时，可以将其存储指派给一个或多个数据存储设备（请参见《系统管理指南，卷 1》中的第 7 章“初始化数据库设备”）。有关这些设备的信息存储在 master.dbo.sysdevices 中。声明要用于数据库的设备，以及此数据库使用每个设备的程度。数据库可以占据设备上的所有可用空间，或者其它数据库可以共享设备上的空间，或者这二者的任何组合。段（数据库中的存储的逻辑分组）允许在逻辑上或物理上将一些数据与其它数据分开保存。例如，为帮助进行灾难恢复， Sybase 强烈建议在物理上将事务日志与数据库中的其它数据分开。

逻辑和物理数据分组可帮助数据库更好地执行。例如，可以为您知道将随时间增大的数据集保留数据库的一部分，方法是仅为此数据集指派特定段。也可以在物理上将大量使用的索引与其数据分开以帮助防止磁盘“失败”，这会延长读取和写入响应时间。

注释　对于 Adaptive Server，设备提供了数据库到物理存储的逻辑映射，而段提供了数据库对象到设备的逻辑映射。若要实现空间分配目标，了解这些逻辑层之间的相互作用很重要。

每个数据库最多可以具有 32 个命名段。Adaptive Server 创建并使用其中三个段：

- system 段 — 包含大多数系统目录。
- default 段 — 如果未在创建对象时指定段，则使用该段。
- logsegment — 存储事务日志。

可以在 system 段中存储用户表，但 logsegment 完全为日志而保留。

Adaptive Server 在 master.dbo.sysusages 中跟踪每个数据库的各区段。sysusages 中的每个条目说明数据库的一个片段。片段是连续一组逻辑页，全部位于同一设备上，允许存储同一组段。片段又称为“磁盘区段”。

由于 Adaptive Server 分配和维护数据库空间的方式所致，这些磁盘片段是 256 个逻辑页 （这是一个**分配单元**）的偶数倍。在决定设备多大时，请考虑所需的分配单元数，因为设备大小应可被分配单元大小（逻辑页大小的 256 倍）整除。否则，将浪费该设备末尾的空间，因为 Adaptive Server 无法分配它。例如，如果服务器使用 16K 页，则一个分配单元为 4MB （16K 乘以 256）。如果在该服务器上创建 103MB 的设备，则将无法分配最后的 3 MB，从而浪费了这 3 MB。

注释 主设备是此规则的一种例外。主设备是您安装新服务器时创建的第一个设备。Adaptive Server 保留主设备开头的 8K 空间作为不属于任何数据库的配置区域。在创建主设备时考虑此空间。 8K 是 0.0078125MB（约 0.008MB）。例如，如果指定 200.008MB 而非 200MB 作为大小，将在主设备中浪费最少的空间。

数据库不能大于 2,147,483,648 个页。逻辑页大小决定字节数：使用 2K 页，它是 4 TB，在 16K 页 Adaptive Server 上，它是 32 TB。

可以采用所需任何方式在设备之间划分数据库存储。每个数据库的磁盘片段数的理论限制为 8,388,688。不过，实际限制取决于 Adaptive Server 内存配置。若要使用数据库， Adaptive Server 必须在内存中保留数据库的存储说明。这包括数据库的 “磁盘镜像”的说明，磁盘镜像包括您为数据库将存储指派到的所有磁盘片段。实际上，数据库的存储复杂程度受为 Adaptive Server 配置的内存量的限制，并且通常不是问题。

不过，如果数据库的磁盘镜像包含数以千计的磁盘片段，其性能可能受损。当 Adaptive Server 需要读取或写入页时，它通过在磁盘镜像中查找该页来将该页的逻辑页码转换为磁盘上的位置。虽然此查找很快，但它确实需要时间，并且随着您向镜像中添加更多磁盘片段，时间变得更长。

第 2 章

网络和性能

本章讨论在使用 Adaptive Server 时，网络在应用程序的性能方面所起的作用。

通常，系统管理员是第一个察觉到网络或性能问题的人，这些问题包括

- 进程响应时间在无明显原因的情况下发生显著变化。
- 返回大量行的查询所用的时间比预期的要长。
- 在正常的 Adaptive Server 处理期间，操作系统处理速度变慢。
- Adaptive Server 在特定的操作系统处理期间，处理速度变慢。
- 某一特定的客户端进程似乎使所有其它进程都变慢。

潜在的性能问题

可能由网络导致的一些基本问题包括：

- Adaptive Server 使用网络服务的效率低下。
- 已达到了网络的物理限制。
- 进程检索不需要的数据值，这无谓地增加了网络通信量。
- 进程过于频繁地打开和关闭连接，从而增加了网络负担。

- 进程频繁地提交同一 SQL 事务，从而导致网络通信量过多、过剩。
- Adaptive Server 没有足够的网络内存。
- Adaptive Server 网络包大小不够大，无法处理某些客户端所需类型的处理。

关于网络性能的基本问题

当研究与网络有关的问题时，可考虑下列问题：

- 哪些进程经常检索大量数据？
- 是否发生了大量的网络错误？
- 网络的总体性能如何？
- 利用 SQL 和存储过程执行的事务有哪些？
- 是否有大量进程使用两阶段提交协议？
- 是否正在通过网络进行复制服务？
- 操作系统正在使用的网络处理有多少？

技术摘要

收集完数据后，可利用多种应该能够改善网络性能的技术，其中包括：

- 对多数数据库活动都使用小包
- 对进行大量数据传送的任务使用较大的包大小
- 使用存储过程减少总体通信量
- 过滤数据以避免大批量的传送
- 将重网络负荷用户与普通用户隔离
- 对特殊情况使用客户端控制机制

更改网络配置以观察对性能产生的影响时可使用 sp_sysmon。请参见《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

引擎和线程密切连接

配置为线程化模式时， Adaptive Server 任务与特定引擎具有有限的密切连接。

网络监听器

网络监听器是一种系统任务，它在指定的网络端口上监听进入的客户端连接，并为每个客户端连接创建一个数据库管理系统任务。 Adaptive Server 为每个 Adaptive Server （对进入的客户端连接请求）要进行监听的网络端口创建一个监听器任务。最初，这些端口包括 interfaces 文件中的 master 条目。

网络监听器任务的初始数目和接口文件中主项的数目相等。网络监听器的最大数目 （包括启动时创建的网络监听器）是 32。例如，如果启动时服务器名称下的 interfaces 文件中的 master 条目数为 2，则可以再创建 30 个监听器任务。

创建的每个附加的监听器所耗费的资源与一个用户连接占用的资源相同。因此，每创建一个网络监听器， Adaptive Server 能接收的用户连接就要减少一个。 number of user connections 配置参数不仅包括网络监听器的数目，还包括附加的监听器端口的数目。

监听器端口的数目是启动时根据接口文件中的主项数决定的。

有关 interfaces 文件的详细信息，请参见 《系统管理指南，卷 1》中的第 1 章 “系统管理概述”。

进程模式中的网络监听器

在进程模式中，每个 Adaptive Server 引擎是单独的进程；因此，当 Adaptive Server 是单个进程时，网络监听器运行的方式与在线程化模式中时略有不同。

在进程模式中：

- Adaptive Server 对每个端口使用一个监听器任务。通过从一个引擎切换到另一个引擎来平衡负载，每个监听器任务都起到多个逻辑监听器的作用。例如，一个 64 引擎、带有两个主端口的 Adaptive Server 有两个监听器任务，而这两个监听器任务充当 128 个逻辑监听器任务，因此此服务器具有两个物理监听器和 128 个逻辑监听器。除非端口上还没有监听器，否则启动引擎 3 上的监听器并不能使 Adaptive Server 生成新的监听器任务

- 监听器任务在启用了此监听器的引擎上接收连接。因此单个监听器任务对应于多个逻辑监听器。
- 停止特定引擎上的监听器会导致终止该引擎的逻辑监听器，原因是监听器任务不再切换到此引擎。 Adaptive Server 在当此引擎是允许操作的最后一个引擎时终止监听器任务。

Adaptive Server 使用网络的方式

所有客户端 / 服务器通信都是通过包经由网络进行。包中包含标头和路由信息，以及所携带的数据。

客户端启动到服务器的连接。该连接发送客户端请求及服务器响应。应用程序可根据执行的请求任务同时打开任意多个连接。

客户端和服务器之间使用的协议称为 Tabular Data Stream™ (TDS)，它构成了许多 Sybase 产品的通信基础。

配置 I/O 控制器

Adaptive Server 包括 I/O 控制器和 I/O 控制器管理器。

I/O 控制器发出、跟踪、轮询和完成 I/O。每种 Adaptive Server I/O 类型 — 磁盘、网络、 Client-Library 以及 CIPC （对于 Cluster Edition）— 都有自己的 I/O 控制器。

Adaptive Server 可以包括磁盘或网络控制器的多个实例 （不允许多个 CIPC 或 Client-Library 控制器）。每个任务表示一个控制器。例如，配置三个网络任务意味着网络 I/O 使用三个控制器。

为每个控制器任务分配了专用的操作系统线程。附加任务意味着更多 CPU 资源专用于轮询和完成 I/O。

每个系统一个 I/O 控制器通常足够了。不过，在 I/O 率非常高或单线程性能很低的系统上，可能需要附加控制器。在这种情况下，引擎可能变得急需 I/O，并且吞吐量降低。

使用 sp_sysmon “Kernel Utilization”部分确定是否需要附加 I/O 任务。

在以下情况下，考虑附加 I/O 任务：

- I/O 任务的“Thread Utilization (OS %)”超过“Engine Busy Utilization”。
- I/O 任务的 “Thread Utilization (OS %)”超过 50%。
- Controller Activity 部分中返回 “max events”的轮询大于零。
- Controller Activity 部分中的 “average events per poll”大于三。

您为 Adaptive Server 配置的模式决定它如何处理 I/O。在线程化模式（缺省值）中， Adaptive Server 对 I/O 使用线程化轮询；在进程模式中，Adaptive Server 对 I/O 使用轮询计划程序。

在进程模式中， Adaptive Server 为每个引擎指派自己的网络、磁盘和 Open Client 控制器。在计划程序轮询 I/O 时，它仅搜索引擎的本地控制器 （除了 CIPC 外，其所有引擎共享单个控制器）。

进程模式轮询的一项好处是，在缩放引擎数量时，缩放可用于管理 I/O 的 CPU 量 （即，更多引擎 = 更多 CPU）。不过，可以配置过多 CPU 来管理 I/O，从而将更多时间用于高于执行任务所需的引擎数。其它性能影响是启动 I/O 的引擎必须完成 I/O （即，如果引擎 2 上运行的任务发出磁盘 I/O，则该 I/O 必须由引擎 2 完成，即使其它引擎空闲也是如此）。这意味着在有要执行的任务时，引擎也可能保持空闲，并且如果负责的引擎正在运行 CPU 密集型任务，则 I/O 可能引发其它延迟。

在配置为线程化轮询时，控制器管理器为每个控制器指派任务，并将此任务放入 syb_system_pool 中。因为 syb_system_pool 是专用池，所以它创建线程来为每个任务服务。此线程以独占方式为 I/O 控制器运行轮询和完成例程。因为此线程专用于执行此任务，所以此任务可阻止等待 I/O 完成，从而减少了系统呼叫和空轮询的数量。

可以创建多个线程来为每个 I/O 控制器服务，从而可以避免单线程饱和问题，出现此类问题时，单个线程无法保持高 I/O 率。

当 I/O 在操作系统级别上完成时，进程模式轮询引入 I/O 延迟。不过，引擎没有检测到 I/O 延迟，因为引擎正在运行其它任务。线程化模式轮询消除了此延迟，因为 I/O 线程任务立即处理完成，并且任何 I/O 延迟是设备的功能，不受查询线程执行置于系统上的 CPU 负载的影响。

在线程化模式中，查询处理器和用户任务在完成计划程序时无需为 I/O 轮询进行环境切换。线程化轮询减少了轮询所花时间长度 （以所有线程的总 CPU 时间的百分比为单位），从而使 Adaptive Server 在 CPU 消耗方面更高效。

使用具有 number of disk tasks 和 number of network taks 的 sp_configure 来确定专用于处理 I/O 的任务数量和任务使用的线程轮询方法。

请参见 《系统管理指南，卷 1》中的第 5 章 “设置配置参数”。

缺省情况下，每个 I/O 任务使用 syb_system_pool 中的线程，从而允许在 I/O 轮询期间阻止任务，进而减少忙轮询的开销。在低 I/O 负载期间，这些线程消耗很少的物理 CPU 时间。I/O 线程的 CPU 时间随 I/O 负载的增加而增加，但负载增加量取决于处理器性能和 I/O 实现。

动态重新配置 I/O 任务

在增加 I/O 任务数时，在 Adaptive Server 跨任务平衡负载之前可能有少许滞后。在增加磁盘 I/O 数时， Adaptive Server 快速跨控制器平衡分布。不过，网络任务具有连接和任务之间的关联，因此在增加网络任务数时，不会跨新增加的任务数重新平衡它们。 Adaptive Server 在现有连接断开和进行新连接时重新平衡负载。

必须重新启动 Adaptive Server 才能减少 I/O 任务数。

更改网络包大小

通常， OLTP 发送和接收大量包，但其中只包含非常少的数据。典型的 insert 或 update 语句可能只有 100 或 200 个字节。对某次数据检索来说，即使它连接了多个表，也可能只返回一行或两行数据，仍未完全填满一个包。使用存储过程和游标的应用程序通常也只发送和接收小包。

决策支持应用程序经常包括大批量的查询并返回较大的结果集。

在 OLTP 和 DSS 这两种环境下，可能有可从较大的包中受益的特殊需求（如批处理数据装载或文本处理）。

对于大多数应用程序，缺省的网络包大小 2048 工作良好。如果应用程序仅使用短查询并接收小结果集，请将 default network packet size 更改为 512。

《系统管理指南，卷 1》中的第 5 章 “设置配置参数”说明了如何更改这些配置参数：

- default network packet size
- max network packet size 和 additional network memory 为大包连接提供额外的内存空间

只有系统管理员才能更改这些配置参数。

在用户连接中使用大包或缺省包尺寸

Adaptive Server 为所有配置的用户连接保留了足够的空间，以便使用缺省包大小登录。大的网络包无法使用该空间。使用缺省网络包大小的连接始终具有三个为连接保留的缓冲区。

请求大包的连接将从 additional network memory 区域获得用于其网络 I/O 缓冲区的空间。如果此区域没有足够的空间而无法以大的包大小分配三个缓冲区，连接将用缺省包大小代替。

包的数目至关重要

通常，待传送的包的数目要比包的大小更重要。网络性能包括 CPU 和操作系统处理网络包所需的时间。这种逐包开销对性能的影响最大。较大的包将减少总体开销，达到更高的物理吞吐量 （假设有足够多的数据要发送）。

下面的大传送源可能会从大包中受益：

- 批量复制
- readtext 和 writetext 命令
- 结果集较大的 select 语句
- 使用较大行大小的插入

始终存在这样一点，在该点上，即使增加包大小也不会改善性能，事实上可能会降低性能，因为包并非总是满的。尽管有多种用于预测该点的分析方法，但通常都是通过实验改变大小来得出结果。如果在某时间段和各种条件下进行此类试验，即可确定适用于许多进程的包大小。不过，由于可以为每个连接自定义包大小，您可能还希望为特定进程进行特定试验。

在不同的应用程序之间，结果可能会明显不同。您可能发现批量复制最适合一个包大小，而大图像数据检索在包大小不同时执行得更好。

如果测试表明某些应用程序使用较大包大小时可达到更好的性能，而多数应用程序却发送和接收小包，则客户端请求较大的包大小。

Adaptive Server 评估工具

sp_monitor 系统过程报告包活动。此报告只显示与包相关的输出：

```
...
packets received   packets sent   packet errors
----------------   ------------   --------------
10866(10580)       19991(19748)              0(0)
...
```

您也可使用这些全局变量：

- *@@pack_sent* — Adaptive Server 发送的包数。
- *@@pack_received* — 接收的包数。
- *@@packet_errors* — 错误数。

这些 SQL 语句表明如何使用这些计数器：

```
select "before" = @@pack_sent
select * from titles
select "after" = @@pack_sent
```

sp_monitor 和全局变量都报告自上次重新启动 Adaptive Server 以来所有用户的所有包活动。

有关 sp_monitor 和这些全局变量的详细信息，请参见 《Transact-SQL 用户指南》中的第 14 章 “使用批处理和控制流语言”。

其它评估工具

操作系统命令还提供有关包传送的信息。请参见操作系统文档。

用来减少网络通信量的基于服务器的技术

利用存储过程、视图及触发器可减少网络通信量。这些 Transact-SQL 工具可在服务器存储大量代码，从而只有一些简短的命令需要通过网络发送。

- 存储过程 — 如果 SQL 转换为存储过程，则发送大批量 Transact-SQL 命令的应用程序可减轻网络的负担。视图也可帮助减少网络通信量。

  通过关闭 doneinproc 包也能减少网络开销。

- 只请求需要的信息 — 应用程序应只请求需要的行和列，从而在服务器上过滤掉尽可能多的数据以减少需要发送的包数。在许多情况下，这还可减少磁盘 I/O 负载。

- 大批量传送　— 同时降低总体吞吐量并增加平均响应时间。如果可能，在非高峰时间执行大批量传送。如果大批量传送很常见，请考虑获得适合此类传送的网络硬件。表 2-1 显示了某些网络类型的特性。

表 2-1：网络选项

类型	特性
令牌环	在高峰使用期间，令牌环硬件的响应要比以太网硬件好。
光纤	光纤硬件提供非常宽的带宽，但它通常特别昂贵，难以在整个网络中使用。
单独的网络	使用单独的网络来处理最高容量工作站与 Adaptive Server 之间的通信量。

- 网络过载 — 在数据库用户开始向其系统管理员抱怨之前，网络管理者很少能觉察到问题。

  当本地网络管理者考虑添加资源时，应准备好向其提供预测的或实际的网络需求。另外，监控网络并尝试预计新添加的设备或应用程序需求可能导致的问题。

其它服务器活动的影响

您应该了解其它服务器的活动与维护对网络所产生的影响，尤其是：

- 两阶段提交协议
- 复制处理
- 备份处理

这些活动（尤其是复制处理和两阶段提交协议）涉及网络通信。大量使用这些活动的系统可能会经历与网络相关的问题。因此，只能根据需要尝试执行这些活动。当其它网络活动速度缓慢时应尽量限制备份活动的次数。

单用户与多用户

在试图解决数据库问题之前，必须考虑到其它用户的存在，尤其是当那些用户正使用同一个网络的时候。

由于多数网络一次只能传送一个包，所以当进行大的传送时，可能有许多用户被延迟。此类延迟可能导致锁被持有的时间变长，这又将导致更长的延迟。

如果响应时间异常得长而常规测试指示没有问题，则此情况可能是由于同一网络上的其他用户导致的。在这种情况下，可询问用户进程是何时开始运行的，操作系统是否变得反应迟缓，其它用户是否正在进行大的传送等。

通常情况下，在发掘数据库系统的深层次原因来解决异常响应时间问题之前，首先应考虑对多用户的影响，如由长期事务所导致的延迟。

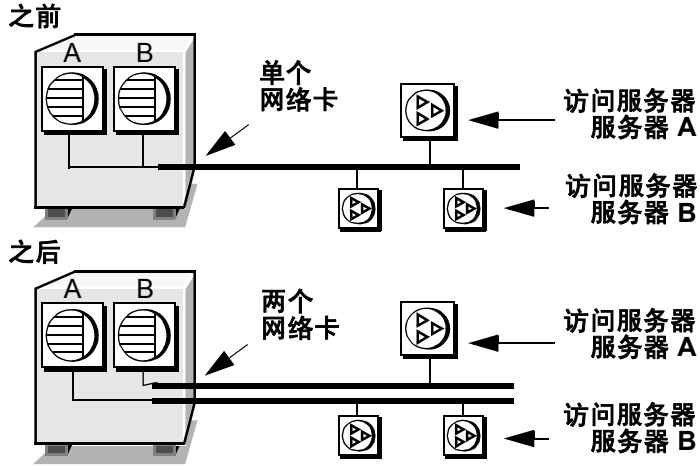
改善网络性能

可以采用多种方法改善网络性能。

隔离重网络负荷用户

通过将重网络负荷用户置于单独的网络中，使其与普通网络用户隔离，如图 2-1 所示。

图 2-1：隔离重网络负荷用户



在“之前”图中，访问两个不同的 Adaptive Server 的客户端使用一个网卡。访问服务器 A 和 B 的客户端必须争用网络并通过该网卡。

在“之后”一图中，访问服务器 A 的客户端使用一个网卡，访问服务器 B 的客户端使用另一个网卡。

在 TCP 网络中设置 *tcp no delay*

缺省情况下， tcp no delay 设置为 on，这意味着禁用包批处理。

当 tcp no delay 设置为 off 时，网络对包进行批处理，从而暂时延迟了通过网络分派部分包。虽然这可以在终端模拟环境中改善网络性能，但它可能会降低发送和接收小批处理的 Adaptive Server 应用程序的性能。若要启用包批处理，请将 tcp no delay 设置为 0 或 off。

配置多个网络监听器

对单个 Adaptive Server 使用两个（或更多）端口监听。通过设置环境变量 DSQUERY，可将前端软件指向任意配置的网络端口。

使用多个网络端口可分散网络负担，消除或减少网络瓶颈，从而增加 Adaptive Server 吞吐量。

有关配置多个网络监听器的详细信息，请参见针对所用平台的《Adaptive Server 配置指南》。

第 3 章

使用引擎和 CPU

Adaptive Server 多线程体系结构是为满足单处理器和多处理器系统中的高性能而设计的。本章描述 Adaptive Server 如何使用引擎和 CPU 来执行客户端请求和管理内部操作。它介绍了 Adaptive Server 对 CPU 资源的使用，描述了 Adaptive Server 对称多重处理 (SMP) 模式，并通过一个处理方案演示了任务调度。

本章还提供了多处理器应用程序设计的原则，并描述了如何评估和调优与 CPU 和引擎相关的功能。

背景概念

关系数据库管理系统 (RDBMS) 必须能够响应许多并发用户的请求。 RDBMS 还必须保持其事务状态，同时确保所有事务属性。Adaptive Server 基于多线程、单进程的体系结构，该体系结构可管理数以千计的客户端连接和多个并发客户端请求，而不会使操作系统超负荷。

在具有多个 CPU 的系统中，可通过将 Adaptive Server 配置为使用多个 Adaptive Server 引擎来提高性能。在线程化内核模式 （缺省值）中，每个引擎是一个操作系统线程。在进程模式中，每个引擎是一个单独的操作系统进程。

所有引擎都是对等的，当它们在公共用户数据库和内部结构（如数据高速缓存和锁链）上操作时，通过共享内存进行通信。 Adaptive Server 引擎为客户端请求提供服务。它们执行所有数据库功能，包括搜索数据高速缓存、发出磁盘 I/O 读写请求、请求和释放锁、更新以及记录日志。

Adaptive Server 管理在处理客户端请求的引擎之间共享 CPU 资源的方式。它也管理影响处理资源的系统维护（如，数据库锁定、磁盘 I/O 和网络 I/O）。

Adaptive Server 如何处理客户端请求

Adaptive Server 为每个新连接创建一个新的客户端任务。以下是它执行客户端请求的方式：

1 客户端程序建立与 Adaptive Server 的网络套接字连接。

2 Adaptive Server 从在启动时分配的任务池中指派任务。该任务由 Adaptive Server 进程标识符或 spid（在 sysprocesses 系统表中跟踪它）标识。

3 Adaptive Server 将客户端请求的环境（包括权限和当前数据库等信息）传送给任务。

4 Adaptive Server 对请求进行分析、优化和编译。

5 如果已启用并行查询执行，则 Adaptive Server 分配子任务以帮助完成并行查询执行。子任务称为工作进程，《性能和调优系列：查询处理和抽象计划》中讨论了相关内容。

6 Adaptive Server 执行任务。如果该查询是被并行执行的，则任务将合并各子任务的结果。

7 任务使用 TDS 信息包将结果返回给客户端。

Adaptive Server 为每个新用户连接分配专用的数据存储区域、专用堆栈以及其它内部数据结构。

Adaptive Server 使用堆栈跟踪处理过程中每个客户端任务的状态，并使用同步机制（如排队、锁定、信号和螺旋锁）来确保一次只有一个任务可以访问任何公共的、可修改的数据结构。必须采用这些机制，因为 Adaptive Server 并发处理多个查询。没有这些机制，如果两个或更多查询要访问相同数据，则会损坏数据完整性。

这些用于环境切换开销的数据结构需要最少的内存资源和系统资源。它们中有些是面向连接的并包含客户端的静态信息。

其它数据结构是面向命令的。例如，当客户端向 Adaptive Server 发送命令时，可执行查询计划存储在内部数据结构中。

客户端任务实现

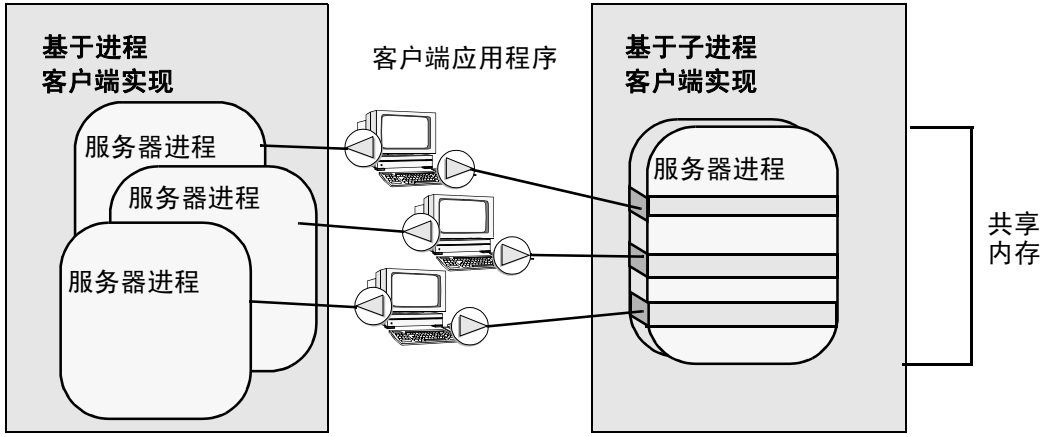
Adaptive Server 客户端任务作为子进程或“轻量进程”而不是操作系统进程来实现。子进程仅使用进程所使用资源的一小部分。

多个并发执行的进程比多个子进程需要更多的内存和 CPU 时间。进程也需要操作系统资源，以将环境从一个进程切换到另一个。

使用子进程消除了大部分与“每个连接一个进程”体系结构相关的分页、环境切换、锁定和其它操作系统功能的开销。子进程启动后不需要操作系统资源，并可共享许多系统资源和结构。

图 3-1 说明了作为进程实现的客户端连接和作为子进程实现的客户端连接所需系统资源的差异。子进程存在单个执行程序进程实例中并在其中操作，其地址空间位于共享内存中。

图 3-1：进程与子进程体系结构



若要使 Adaptive Server 获得最大的处理能力，请在数据库计算机上仅运行必要的非 Adaptive Server 进程。

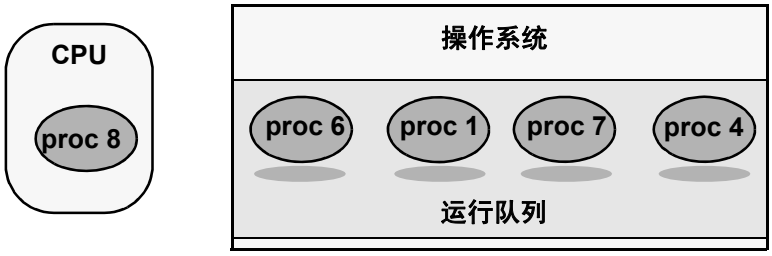
单 CPU 进程模式

在单 CPU 系统中，Adaptive Server 作为单个进程运行，按照操作系统的调度与其它进程共享 CPU 时间。

将引擎调度给 CPU

图 3-2 显示了单 CPU 环境的运行队列，其中进程 8 (proc 8) 正在 CPU 上运行，而进程 6、1、7 和 4 正在操作系统运行队列中等待 CPU 时间。进程 7 是一个 Adaptive Server 进程；其它进程可以是任何操作系统进程。

图 3-2：排队等待单 CPU 的进程

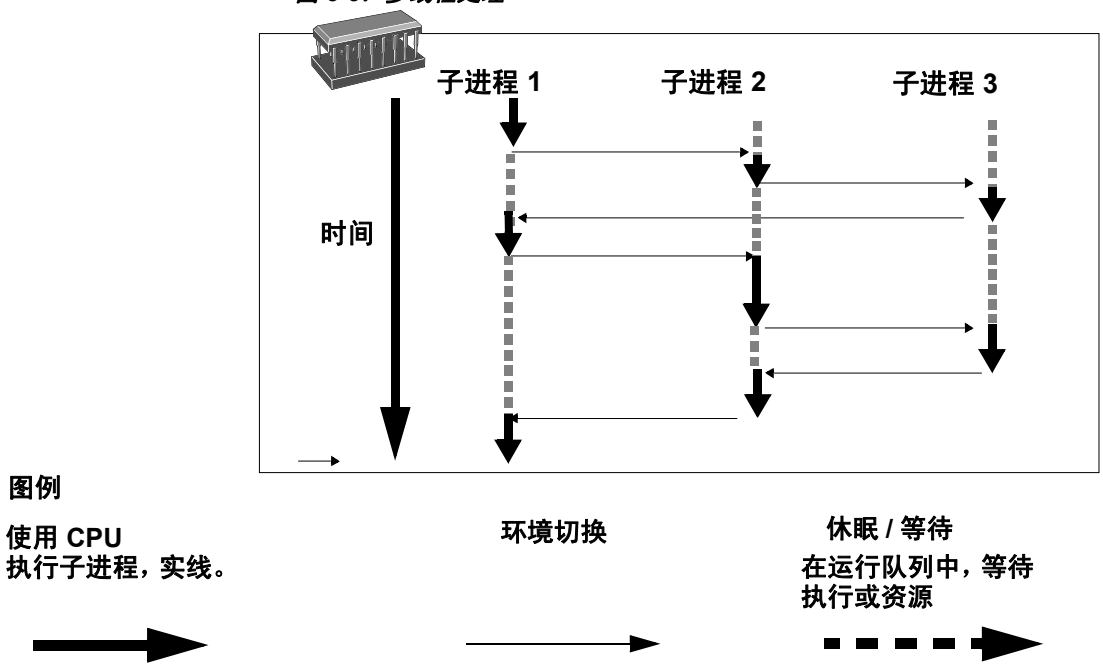


在多任务环境中，多个进程或子进程并发执行，交替共享 CPU 资源。

图 3-3 显示了多任务环境中的三个子进程。各子进程通过随时切换到和断开引擎来共享单 CPU。

在任何同一时间内只执行一个进程。其它进程在各进程阶段中休眠。

图 3-3：多线程处理



将任务调度给引擎

图 3-4 显示了在单 CPU 环境中排队等候 Adaptive Server 引擎的任务（或工作进程）的内部处理。 Adaptive Server 而不是操作系统，动态地将客户端任务从运行队列调度到引擎。引擎处理完一项任务后，它执行位于运行队列开头的任务。

任务开始运行后，引擎将一直处理它直到发生下列事件之一：

- 任务完成、将数据 （如果有）、元数据和状态返回到客户端。当任务完成后，它在 sp_sysmon 部分 `Task Context Switches Due To` 中显示为 `Network Packet Received`。
- 如果 Adaptive Server 引擎未找到任何可运行的任务，它可以将 CPU 归还给操作系统，或按 runnable process search count 设置的次数进行循环，继续查找要运行的任务。

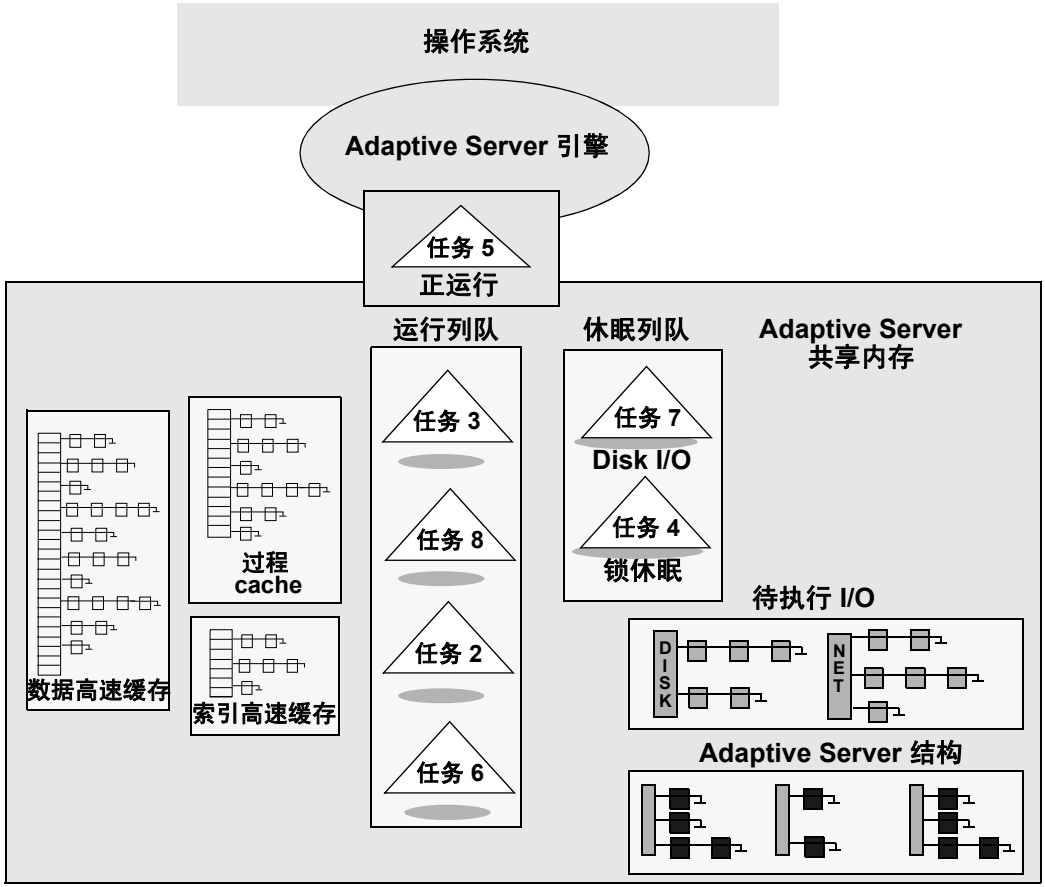
  Adaptive Server 引擎尝试尽可能长地保留处理器的调度。引擎会一直运行，直到操作系统清空它为止。不过，如果可用工作不足，引擎会检查 I/O 和可运行任务。

- 任务已运行了可配置的时间段，并达到屈服点（sp_sysmon 中的 Voluntary Yields）。任务放弃引擎，队列中的下一进程开始运行。第 33 页的 “调度客户端任务处理时间”详细论述了此项工作。

在具有多个活动任务的单 CPU 系统上执行 sp_who 时，sp_who 输出仅将一个任务显示为 “running” — 它是 sp_who 任务本身。运行队列中所有其它任务的状态都为 “runnable”。 sp_who 输出还显示所有休眠任务的原因。

图 3-4 还显示了具有两个休眠任务的休眠队列，以及共享内存中的其它对象。任务在等待资源或磁盘 I/O 操作的结果时处于休眠状态。

图 3-4：任务排队等待 Adaptive Server 引擎



执行任务调度

调度程序管理客户端任务和内部管家事务的处理时间。

调度客户端任务处理时间

time slice 配置参数防止执行任务在执行过程中独占引擎。调度程序允许任务在 Adaptive Server 引擎上执行的最长时间等于 time slice 和 cpu grace time 的值之和，使用 time slice 的缺省时间 （100 毫秒， 1/10 秒或等同于一个时钟周期）和 cpu grace time 的缺省时间（500 个时钟周期或 50 秒）。

Adaptive Server 调度程序不强制任务从 Adaptive Server 引擎上断开。如果任务未持有重要资源 （如螺旋锁），则在屈服点处主动放弃引擎。

每次任务达到屈服点时，它将检查是否超过 time slice。如果它还未超过，则任务继续执行。如果执行时间确实超过 time slice，则任务主动放弃引擎。不过，如果任务在超过 time slice 后未放弃，则 Adaptive Server 会在它超过 cpu grace time 后终止该任务。任务不放弃的最常见原因是系统调用没有及时返回。

有关使用 sp_sysmon 确定任务主动放弃的次数的详细信息，请参见第 31 页的 “将任务调度给引擎”。

若要增加 CPU 密集型应用程序在放弃之前在引擎上运行的时间，请为特定的登录、应用程序或存储过程指派执行属性。

如果任务在完成客户端请求之前必须放弃引擎，该任务将转到运行队列的末尾，除非运行队列中没有其它任务。如果在宽限期间，执行任务达到屈服点时队列中没有任务，则 Adaptive Server 将授予该任务另一处理间隔。

通常，任务在 cpu grace time 间隔结束之前在屈服点放弃引擎。任务可能遇不到屈服点而超出 time slice 间隔。如果 time slice 设置得太低，引擎可能消耗太多时间于任务间切换，这往往会增加响应时间。如果 time slice 设置得太高， CPU 密集型进程可能会独占 CPU，这将延长短任务的响应时间。如果应用程序遇到时间片错误，则调整 time slice 的值将无效，但调整 cpu grace time 的值将有效。不过，请在调整 cpu grace time 的值之前研究 time slice 错误的原因。您可能需要与 Sybase 技术支持部门联系。

当 cpu grace time 结束时， Adaptive Server 因时间片错误终止任务。如果收到时间片错误，请尝试将 cpu grace time 的时间增加为当前时间的四倍。如果此问题仍然存在，请打电话与 Sybase 技术支持部门联系。

请参见第 4 章 “分配引擎资源”。

保持空闲时间段 CPU 可用性

create thread pool 和 alter thread pool 的 idle timout 参数确定此池中的线程将在查找工作多少微秒后进入休眠状态。只能为引擎池而不能为 RTC 池设置 idle timout。请参见 《系统管理指南，卷 1》中的 “设置配置参数”。

idle timout 的缺省值为 100 微秒。不过， Adaptive Server 可能无法精确支持超时期限，尤其当值较低 （低于 100）时。

在为 idle timeout 设置了值后， Adaptive Server 在配置文件中的 Thread Pool 标题下注册该值：

```
[Thread Pool:new_pool]
     number of threads = 1
     idle timeout = 500
```

将 idle timeout 设置为 -1 可阻止引擎放弃。使用此设置，引擎会消耗 100% 的 CPU。

注释 idle timeout 参数取代了 Adaptive Server 15.7 之前的版本中使用的 runnable process search count 配置参数。 idle timeout 只能用于线程化模式。不过， runnable process search count 仍可用于进程模式。

Adaptive Server SMP 进程模式

Adaptive Server 的对称多重处理 (SMP) 实现将 Adaptive Server 的多线程体系结构的性能优势扩展到多处理器系统。在 SMP 环境中，多个 CPU 协同工作的速度比单个处理器所能达到的速度快。

SMP 设计用于具有下列特性的计算机：

- 对称多重处理操作系统
- 通过公用总线共享内存
- 2 至 1024 个处理器 （在进程模式中为 128 个处理器）
- 非常高的吞吐量

将引擎调度给 CPU

SMP 的对称特性是指进程与 CPU 间没有关联 — 进程未连接到特定的 CPU。由于没有 CPU 关联，操作系统将引擎调度到 CPU 的方式与它将非 Adaptive Server 进程调度到 CPU 的方式相同。将任何进程调度到处理器（包括 Adaptive Server 引擎）由操作系统执行，操作系统可以在任何时间抢占 Adaptive Server 引擎来运行任意任务。如果 Adaptive Server 引擎未找到任何可运行的任务，它可以将 CPU 归还给操作系统，或按 idle timeout 参数指定的时间长度继续查找要运行的任务。

在一些情况下，可以通过强制 Adaptive Server 线程与特定 CPU 或一组 CPU 相关联来提高性能。例如，将引擎分组到数量最少的物理套接字中可提高 L2 和 L3 高速缓存命中率，从而提高性能。

在单个套接字对所有引擎和 I/O 线程具有足够的并行度的配置中（例如运行 4 引擎 Adaptive Server 的 8 核套接字），考虑使用 dbcc tune 或操作系统（一般推荐）将 Adaptive Server 引擎绑定到单个套接字。有关将线程或进程绑定到 CPU 的说明，请参见操作系统文档。

将 Adaptive Server 任务调度到引擎

在 SMP 环境中将 Adaptive Server 任务调度到引擎与在单 CPU 环境中调度任务相似，如第 31 页的“将任务调度给引擎”中所述。不过，在 SMP 环境中：

- 每个引擎都有运行队列。任务与引擎间有软密切连接。任务在引擎上运行时，它创建一个到该引擎的密切连接。如果任务放弃该引擎并再次排队，它往往排在同一引擎的运行队列上。
- 任何引擎都可处理全局运行队列中的任务（除非已使用逻辑进程管理将任务指派给特定引擎或引擎集）。
- 引擎查找要执行的任务时，首先会在本地和全局运行队列中查找，然后在其它引擎的运行队列中查找，挪用其中具有适当属性的任务。

多个网络引擎

在进程模式中，用户登录到 Adaptive Server 时，会按循环方式将任务指派给引擎之一，该引擎将成为此任务的网络引擎。此引擎确定包大小、语言、字符集和其它登录设置。任务的所有网络 I/O 都由其网络引擎管理，直到此任务注销。

在线程化模式中，任何引擎都可以为任何任务发出网络 I/O。网络轮询由 syb_system_pool 中的专用网络任务执行。

任务优先级和运行队列

Adaptive Server 可以增加某些任务的优先级，特别是在这些任务持有重要资源或已经不得不等待资源时。另外，逻辑进程管理允许使用 sp_bindexeclass 和相关系统过程给登录、过程或应用程序指派优先级。

有关性能调优和任务优先级的详细信息，请参见第 4 章 “分配引擎资源”。

每个任务都有为其指派的优先级；该优先级可在任务的生存期限内更改。引擎查找要运行的任务时，它首先扫描自己的高优先级队列，然后查找高优先级的全局运行队列。

如果没有高优先级任务，它查找中优先级，然后查找低优先级。如果它在自己的运行队列或全局运行队列上未找到要运行的任务，它可检查另一引擎的运行队列，并挪用另一引擎的任务。优先级、局部和全局队列的组合，以及工作量不均匀时在引擎间移动任务的能力共同保证了负载平衡。

全局或引擎运行队列中的任务都处于可运行状态。任务在任何运行队列中时， sp_who 的输出将任务列为 “runnable”。

处理方案

这些步骤描述了如何在 SMP 环境中调度任务。单处理器系统的执行周期非常相似。单处理器系统用相同的方式处理任务切换、任务等待磁盘或网络 I/O 时将其置为休眠，以及检查队列。

1 在进程模式中，连接登录到 Adaptive Server 时，会将它指派给管理其网络 I/O 的任务。

  任务将连接指派给一个引擎或引擎组并确定包大小、语言、字符集和其它登录设置。任务在等待客户端发送请求时休眠。

2 查找客户端请求。

  在进程模式中，另一个任务在每个时钟周期中检查一次传入的客户端请求。

  在线程化模式中， Adaptive Server 在新请求到达时立即唤醒专用线程。

  当此第二个任务从连接中找到命令 （或查询）时，它唤醒第一个任务并将其放置在其运行队列的末尾。

3　执行客户端请求。

任务成为队列中的第一位时，查询处理器分析、编译并开始执行任务查询计划中定义的步骤。

4　执行磁盘 I/O。

如果任务需要访问被另一用户锁定的页，系统会将其置为休眠直到该页可用。如此等待后，该任务的优先级提高，并将其放置在全局运行队列中，以便任何引擎都可运行它。

5　执行网络 I/O。

在线程化模式中，因为没有网络关联，所以任务从任何引擎返回结果。

在进程模式中，如果任务在其网络引擎上执行，则返回结果。如果任务在其网络引擎之外的引擎上执行，则执行引擎会将该任务添加到网络引擎的高优先级队列中。

异步日志服务

异步日志服务 (ALS) 使 Adaptive Server 获得了可伸缩性，可为高端的对称多处理器系统的日志记录子系统提供更高的吞吐量。

对于启用了 ALS 的每个数据库，一个引擎主要执行日志 I/O，因此仅当日志信号的争用高于一个引擎的处理能力时， ALS 才有用。

如果引擎数少于 4 个，则不能使用 ALS。

启用 ALS

使用 sp_dboption 启用、禁用或配置 ALS

```
sp_dboption database_name, "async log service", "true|false"
```

checkpoint （将所有脏页写入数据库设备）作为 sp_dboption 的一部分自动执行。

此示例为 mydb 启用 ALS：

```
sp_dboption "mydb", "async log service", "true"
```

禁用 ALS

在禁用 ALS 前，请确保数据库中没有活动用户。如果有，您会收到错误消息。

此示例禁用 ALS：

```
sp_dboption "mydb", "async log service", "false"
```

显示 ALS　　使用 sp_helpdb 查看是否已在指定数据库中启用 ALS：

```
sp_helpdb "mydb"
name  db_size        owner  dbid   created      durability
status
----   ------------ -----  ----   ------------ ----------
---------------
mydb          3.0 MB    sa     5  July 09, 2010       full
select into/bulkcopy/pllsort, trunc log on chkpt, async log service

device_fragments                  size         usage
created                    free kbytes
-------------------------------  ------------  --------------------
------------------------- ----------------
master                     2.0 MB                          data only
Jul  2 2010  1:59PM                     320
log_disk                   1.0 MB                          log only
Jul  2 2010  1:59PM          not applicable

---------------------------------------------------------------------
log only free kbytes = 1018
device      segment
--------    ----------
log_disk    logsegment
master      default
master       system
```

理解用户日志高速缓存 (ULC) 的体系结构

Adaptive Server 的日志记录体系结构的特点是使用了用户日志高速缓存 (ULC)，从而使每个任务都拥有自己的日志高速缓存。其它任何任务都无法向此高速缓存写入信息，每当事务生成日志记录时，该任务将继续向用户日志高速缓存写入信息。当提交或中止事务时，或者当用户日志高速缓存已满时，用户日志高速缓存将被刷新到由所有当前任务共享的通用日志高速缓存中，然后再写入到磁盘中。

刷新 ULC 是提交或中止操作的第一步。所需的每个后续步骤都可能导致延迟或增加争用：

1　在最后一页日志页上获得锁。

2　如有必要，分配新的日志页。

3　将日志记录从 ULC 复制到日志高速缓存。

步骤 2 和步骤 3 中的进程要求在最后一页日志页上上锁，这样可以防止任何其它任务向日志高速缓存写入或者执行提交或中止操作。

4　将日志高速缓存刷新到磁盘。

步骤 4 要求重复扫描日志高速缓存，以便对脏缓冲区发出 write 命令。

重复扫描会引起对该日志绑定到的缓冲区高速缓存螺旋锁的争用。在大量的事务负载下，对该螺旋锁的争用现象可能会相当突出。

何时使用 ALS

只要系统运行有 4 个或更多联机引擎，就可对至少有以下一种性能问题的任何数据库启用 ALS：

- 对最后一个日志页的争用现象突出 — sp_sysmon 输出的 “Task Management”报告部分显示一个非常高的值。此示例显示争用的日志页：

```
Task Management            per sec     per xact      count   % of total
-----------------------  ----------  ----------  --------     --------
Log Semaphore Contention       58.0         0.3     34801         73.1
```

- 对日志高速缓存的高速缓存管理器螺旋锁的争用现象突出 — sp_sysmon 输出的有关数据库事务日志高速缓存的 “Data Cache Management”报告部分在 “Spinlock Contention”部分中显示一个高值。例如：

```
Task Management            per sec     per xact      count   % of total
-----------------------  ----------  ----------  --------     --------
Spinlock Contention            n/a         n/a       n/a         40.0
```

- 日志设备中存在未充分利用的带宽。

注释　仅在确定了一个具有很高的事务要求的数据库后再使用 ALS，因为对多个数据库设置 ALS 可能会导致吞吐量和响应时间的意外变化。如果希望对多个数据库配置 ALS，应首先检查吞吐量和响应时间是否令人满意。

使用 ALS

有两个线程将扫描脏缓冲区 （充满尚未写入到磁盘的数据的缓冲区），复制数据，然后将数据写入到日志中。这两个线程是：

- 用户日志高速缓存 (ULC) 刷新器
- 日志写入器。

ULC 刷新器

ULC 刷新器是一个系统任务线程，专门用于将任务的用户日志高速缓存刷新到通用日志高速缓存。当任务准备好提交时，用户向刷新器队列中输入提交请求。每个条目都有一个句柄， ULC 刷新器可通过句柄访问将该请求排入队列的任务的 ULC。 ULC 刷新器任务始终监控着刷新器队列，负责从队列中删除请求并通过将 ULC 页刷新到日志高速缓存来满足请求。

日志写入器

当 ULC 刷新器将 ULC 页刷新到日志高速缓存后，它会将任务请求排入到唤醒队列中。日志写入器将巡查日志高速缓存中的脏缓冲区链，如果发现脏缓冲区则发出写入命令，同时还为那些页面已全部写入到磁盘的任务监控唤醒队列。由于日志写入器巡查脏缓冲区链，因此它知道缓冲区何时可以向磁盘写入。

管家清洗任务可提高 CPU 利用率

管家清洗任务 （sp_who 将其报告为 `HK WASH`）通常作为低优先级任务运行，并且仅在 Adaptive Server 没有要处理的用户任务的空闲周期内运行。运行时，清洗任务会自动将脏缓冲区写入磁盘 （称为自由写入），并执行其它维护任务。这些写入提高了 CPU 利用率并降低了事务处理过程中清洗缓冲区的需求。它们也减少了检查点测试信号的数量和持续时间 — 当检查点进程引起磁盘写入突然急速增加时。

缺省情况下，管家碎片收集以普通用户的优先级操作，它清除逻辑删除的数据并重置行，以使表重新拥有空间。如果 Adaptive Server 配置为线程化模式，请使用 sp_bindexeclass 或 sp_setpsexe 将管家任务设置为较高优先级。

有关管家任务的详细信息和有关重置其优先级的信息，请参见 《系统管理指南，卷 1》中的第 11 章 “诊断系统问题”。

管家清洗任务的副作用

如果管家清洗任务可以刷新所有已配置高速缓存中的全部活动缓冲池，它将唤醒检查点任务。

检查点任务决定了它是否能够对数据库执行检查点操作。如果可以，它将写下检查点日志记录，指明所有脏页已写入磁盘。管家清洗任务所产生的附加检查点可提高数据库恢复的速度。

在反复更新相同数据库页的应用程序中，管家清洗可能会引起一些不必要的数据库写操作。尽管这些写入操作只发生在服务器空闲周期，但在有过载磁盘的系统上可能不会被接受。

配置管家清洗任务

系统管理员可使用 housekeeper free write percent 配置参数控制管家清洗任务的副作用。此参数指定管家清洗任务可增加数据库写操作的最大百分比。有效值为 0 到 100。

缺省情况下， housekeeper free write percent 设置为 1，它允许管家清洗任务继续清洗缓冲区，只要数据库写入增加不超过 1%。在大多数系统上，管家清洗任务在缺省设置下完成的工作使性能及恢复速度得到提高。将 housekeeper free write percent 设置太高可使性能下降。如果要增加该值，请每次只增加 1% 或 2%。

dbcc tune 选项 deviochar 控制管家一次可写入磁盘的批处理大小。

请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》的第 2 章 “使用 sp_sysmon 监控性能”中的 “增加管家批处理限制”。

更改写入可增加的百分比

请使用 sp_configure 更改管家清洗任务可增加数据库写入的百分比：

```
sp_configure "housekeeper free write percent", value
```

例如，若要在数据库写入的频率超过正常 2% 时阻止管家清洗任务运行，请输入：

```
sp_configure "housekeeper free write percent", 2
```

禁用管家清洗任务

禁用管家清洗任务可建立主要运行用户任务的更受控制的环境。若要禁用管家清洗任务，请将 housekeeper free write percent 参数的值设置为 0：

```
sp_configure "housekeeper free write percent", 0
```

没有用于关闭管家杂事任务的配置参数，但可以设置 sp_setpsexe 来降低其优先级。

允许管家清洗任务连续工作

若要允许管家清洗任务在有空闲的 CPU 周期时工作，而不管附加数据库写入的百分比，请将 housekeeper free write percent 参数值设置为 100：

```
sp_configure "housekeeper free write percent", 100
```

请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

测量 CPU 使用率

本节描述了在使用单处理器计算机和使用多处理器计算机上如何测量 CPU 使用率。

单 CPU 计算机

操作系统的 CPU 使用率报告与 Adaptive Server 内部 “CPU 忙”信息总是不对应。

不允许进程模式中的多线程数据库引擎阻止 I/O。执行异步磁盘 I/O 时，Adaptive Server 为其它正等待处理的用户任务提供服务。如果没有要执行的任务， Adaptive Server 进入忙等待循环，等待异步磁盘 I/O 结束。这种低优先级的忙等待循环可导致很高的 CPU 使用率，但由于其优先级很低，它通常是无害的。

线程化模式中的 Adaptive Server 可阻止 I/O。

注释 在进程模式中， Adaptive Server 在执行 I/O 密集型任务时使用很多 CPU 是正常的。

使用 *sp_monitor* 测量 CPU 使用率

使用 sp_monitor 查看 Adaptive Server 在所经历时间间隔中使用 CPU 的时间百分比：

```
last_run                    current_run               seconds
-------------------------   -----------------------   ----------
       Jul 25 2009 5:25PM          Jul 28 2009 5:31PM          360

cpu_busy                io_busy                idle
----------------------- ---------------------- -----------------
5531(359)-99%                           0(0)-0%      178302(0)-0%

packets_received        packets_sent           packet_errors
----------------------- ---------------------- ------------------
57650(3599)                          60893(7252)                0(0)

total_read        total_write      total_errors    connections
----------------- ---------------- --------------- --------------
190284(14095)            160023(6396)              0(0)           178(1)
```

有关 sp_monitor 的详细信息，请参见 《Adaptive Server Enterprise 参考手册》。

使用 *sp_sysmon* 测量 CPU 使用率

sp_sysmon 提供的信息比 sp_monitor 提供的信息详细。sp_sysmon 报告的“Kernel Utilization”部分显示了引擎在采样运行期间的繁忙程度。此输出中的百分比基于将 CPU 分配给 Adaptive Server 的时间；它不是总采样间隔的百分比。

`CPU Yields by Engine` 部分显示在间隔期间引擎将控制权交给操作系统的频率的有关信息。

请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

操作系统命令和 CPU 使用率

用于显示 CPU 使用率的操作系统命令记录在 Adaptive Server 安装和配置指南中。

如果操作系统工具显示多数时间中 CPU 使用率大于 85%，请考虑使用多 CPU 环境或将某些工作卸载到另一 Adaptive Server。

确定何时配置附加引擎

在确定是否要添加附加引擎时，请考虑：

- 现有引擎的负载
- 对资源的争用情况 （如表的锁、磁盘和高速缓存螺旋锁）
- 响应时间

如果现有引擎上的负载超过 80%，增加一个引擎可提高响应时间，除非争用资源情况严重或该附加引擎引起了争用。

配置更多引擎之前，请使用 sp_sysmon 建立基准。在 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》中的 “使用 sp_sysmon 监控性能”中查看 sp_sysmon 输出的以下部分。特别是，研究输出中可能显示争用点的行或部分：

- Logical Lock Contention
- Address Lock Contention
- ULC Semaphore Requests
- Log Semaphore Requests
- Page Splits
- Lock Summary
- Spinlock Contention
- I/Os Delayed by

增加引擎数量后，在相似负载条件下再次运行 sp_sysmon，然后检查报告中的 “Engine Busy Utilization”部分以及上面列出的可能争用点。

将引擎脱机

如果以进程模式运行 Adaptive Server，请使用 sp_engine 使引擎联机和脱机。 Adaptive Server 不接受线程化模式中的 sp_engine。请参见 《Adaptive Server Enterprise 参考手册：过程》中的 sp_engine 条目。

启用引擎到 CPU 的密切连接

缺省情况下，CPU 与 Adaptive Server 中的引擎之间没有密切连接。在高吞吐量环境中，建立引擎到 CPU 的密切连接后性能提高非常轻微。

并非所有操作系统都支持 CPU 关联；在此类系统上，将自动忽略 dbcc tune 命令。每次重新启动 Adaptive Server 时，必须重新发出 dbcc tune。每次打开或关闭 CPU 关联时，Adaptive Server 都将一条消息输出到错误日志中，指出受影响的引擎和 CPU 编号：

```
Engine 1, cpu affinity set to cpu 4.
Engine 1, cpu affinity removed.
```

语法为：

dbcc tune(cpuaffinity, *start_cpu* [, on | off])

start_cpu 指定引擎 0 要绑定到的 CPU。引擎 1 绑定编号为 (*start_cpu* + 1) 的 CPU。确定引擎 *n* 的绑定的公式是：

```
((start_cpu + n) % number_of_cpus
```

有效的 CPU 编号为 0 - CPU 数量减 1。

在有四个 CPU 的计算机 （CPU 编号 0 - 3）和四个引擎的 Adaptive Server 上，以下命令：

```
dbcc tune(cpuaffinity, 2, "on")
```

得出如下结果：

Engine	CPU	
0	2	（指定的 *start_cpu* 编号）
1	3	
2	0	
3	1	

在同一计算机上，使用三个引擎的 Adaptive Server 时，相同命令会产生如下密切连接：

Engine	CPU
0	2
1	3
2	0

Adaptive Server 不使用 CPU 1。

若要禁用 CPU 关联，请使用 -1 代替 *start_cpu*，并指定 off 作为设置：

```
dbcc tune(cpuaffinity, -1, "off")
```

启用 CPU 关联，而无需更改 *start_cpu* 的值，方法是对使用 -1 和 on 作为设置：

```
dbcc tune(cpuaffinity, -1, "on")
```

如果先前未设置 CPU 关联，则 *start_cpu* 的缺省值为 1。

要指定 *start_cpu* 新值，而不更改 on/off 设置，请使用：

```
dbcc tune (cpuaffinity, start_cpu)
```

如果当前已启用 CPU 关联，并且新的 *start_cpu* 与其先前值不同，则 Adaptive Server 会更改每个引擎的密切连接。

如果 CPU 关联关闭，则 Adaptive Server 会记录新的 *start_cpu* 值，并且新关联将在下一次打开 CPU 关联时生效。

若要查看当前值和密切连接是否启用，请使用：

```
dbcc tune(cpuaffinity, -1)
```

此命令只将当前设置输出到错误日志，而不更改关联或设置。

多处理器应用程序设计指南

如果要将应用程序从单 CPU 环境移动到 SMP 环境，本节讨论了一些需要考虑的问题。

多处理器 Adaptive Server 上增加的吞吐量使多个进程更有可能尝试同时访问一个数据页。坚持优良的数据库设计原则以避免争用。下面是一些在 SMP 环境中特别重要的应用程序设计注意事项。

- 多个索引 — 具有多个索引的所有页锁定表更新时，增加的 SMP 吞吐量可能会导致锁争用更加严重。在经常更新的任何表上，不允许超过两个或三个索引。

  有关索引维护对性能的影响的信息，请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

- 管理磁盘 — SMP 的额外处理能力可能增加对磁盘的需求。对于频繁使用的数据库，将数据分布在多个设备上。

  请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

- 调整 create index 命令的 fillfactor — 由于使用多个处理器增加了吞吐量，因此设置较低的 fillfactor 可以临时减少对数据和索引页的争用。
- 事务长度 — 包括多个语句或要很长时间才能运行的事务可能导致锁争用增加。保持事务尽可能短，并在等待与用户交互时避免持有锁 — 特别是排它锁或更新锁。确保基础存储同时提供足够的带宽和足够低的延迟。
- 临时表 — 不会导致争用，因为它们与单个用户相关联并且不共享。不过，如果多个用户进程对临时对象使用 tempdb，则在 tempdb 的系统表上可能会出现某些争用。使用多个临时数据库或 Adaptive Server 版本 15.0.2 及更高版本可缓解对 tempdb 的系统表的争用。

  请参见 《性能和调优系列：物理数据库调优》中的第 7 章 “tempdb 性能问题”。

第 4 章

分配引擎资源

本章介绍如何指派执行属性、 Adaptive Server 如何解释执行属性的组合，以及如何预测各种执行属性指派对系统的影响。

理解有关分配引擎资源的讨论的前提条件是了解 Adaptive Server 如何利用 CPU 资源。有关详细信息，请参见第 3 章“使用引擎和 CPU”。

成功分配资源

Adaptive Server 环境中各执行对象间的交互作用非常复杂。而且，每种环境又各不相同：每种环境均涉及其各自客户端应用程序、登录和存储过程的组合，并因这些实体间的各种相互依赖性而各具特色。

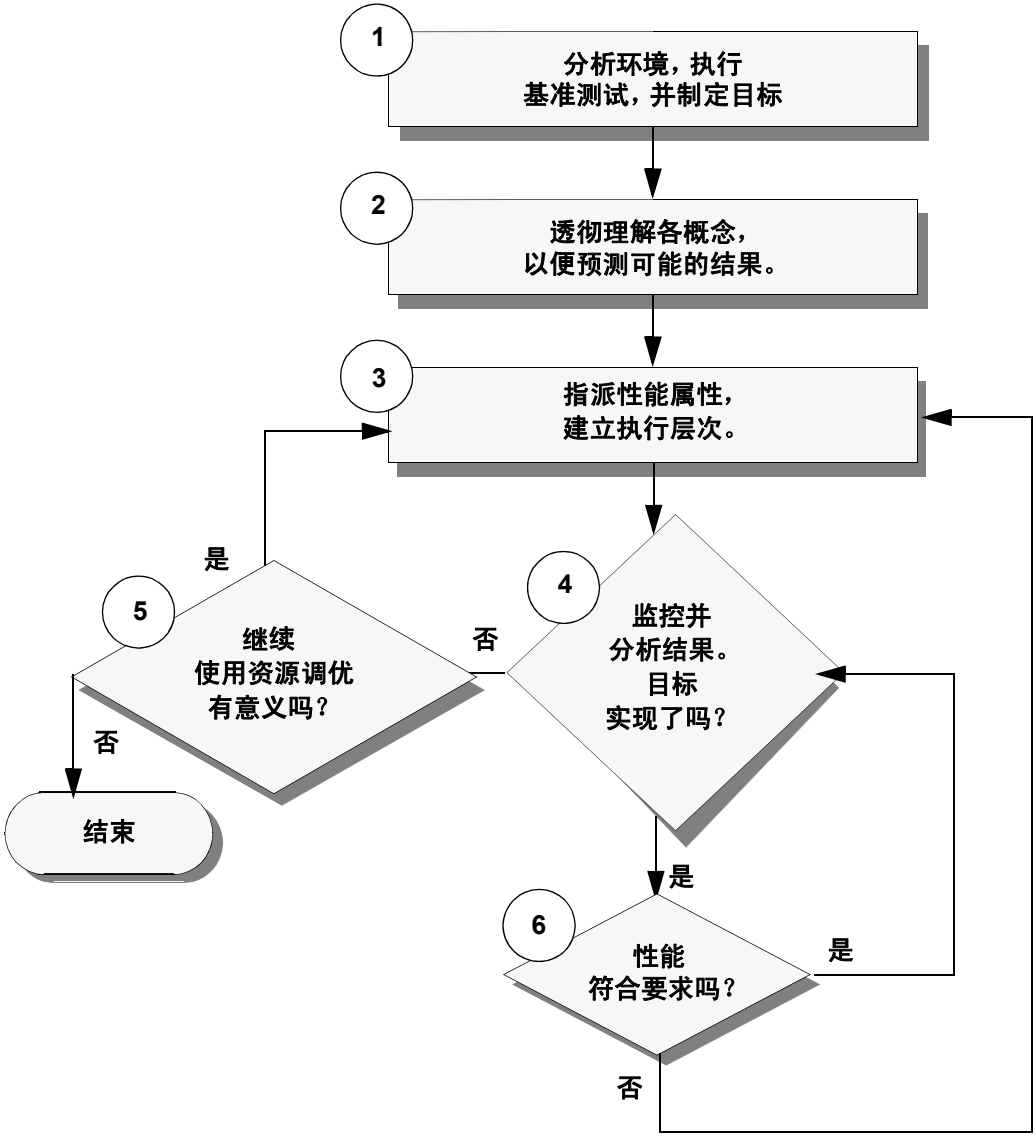
不研究环境及可能的潜在影响便实施执行优先顺序可能会产生意外（而且是负面）的结果。

例如，假定已标识了一个关键的执行对象，并想要提高它的执行属性，以便永久性地或针对单个会话改善性能。如果此执行对象与一个或多个其它执行对象访问同一组表，由于不同优先级的任务之间的锁争用，提升其执行优先级可导致性能下降。

由于每种 Adaptive Server 环境都有其独特性， Sybase 无法提供一个对所有系统都有效的指派执行优先顺序的详细过程。不过，本节会提供指南、试用过程以及常见问题讨论。

图 4-1 显示了指派执行属性所涉及的步骤。

图 4-1：指派执行优先顺序的流程



1　研究 Adaptive Server 环境：

- 分析所有执行对象的行为，尽可能把它们归类。
- 了解各执行对象间的相互依赖性和交互作用。
- 执行基准测试，以便在建立优先顺序后用作比较的基准。
- 考虑在多处理器环境中如何分配处理。
- 确定要提高其性能的关键执行对象。
- 确定允许降低性能的非关键执行对象。
- 为在前两步中确定的执行对象建立一组可计量的性能目标。

请参见第 52 页的 “环境分析与计划”。

2　了解使用执行类的影响：

- 理解与执行类指派相关的基本概念。
- 决定是否创建一个或多个用户定义的执行类。
- 了解不同类级别指派的潜在影响 — 从性能损益和相互依赖性方面看，指派会对环境有怎样的影响？

请参见 n。

3　指派执行类和任何独立的引擎密切连接属性。

4　指派执行优先顺序后，分析运行的 Adaptive Server 环境：

- 运行第 1 步中使用的基准测试，并比较结果。
- 如果未达到预期效果，则进一步考虑执行对象间的交互作用，如第 1 步中所述。
- 研究可能已被忽略的相互关系。

请参见第 54 页的 “结果分析与调优”。

5　按需要多次反复执行第 3 步和第 4 步，以调优结果。

6　随时监控环境。

环境分析与计划

环境分析与计划包括：

- 分析环境
- 执行要用作比较基准的基准测试
- 制定性能目标

分析环境

研究和了解 Adaptive Server 对象如何与环境交互，以便可以对如何实现设置的性能目标做出决策。

分析过程包括两个阶段：

- 第 1 阶段 — 分析每个执行对象的行为。
- 第 2 阶段 — 利用对象分析的结果，预测 Adaptive Server 系统内各执行对象间的交互。

首先，制作一个包含能在此环境中运行的所有执行对象的列表。然后，将每个执行对象及其特性分类。按重要性将彼此相关的执行对象归类。针对每一执行对象，决定下列哪项适用：

- 它是非常关键的执行对象，需要改善响应时间；
- 它是一般重要的执行对象；或
- 它不是关键的执行对象，可以减慢响应时间。

第 1 阶段 — 执行对象行为

典型的分类包括侵入性的 / 未入侵的、 I/O 密集型和 CPU 密集型。例如，将每个对象标识为侵入性的 / 未入侵的、是否属于 I/O 密集型，以及是否属于 CPU 密集型。为获得有用的深入信息，可能需要确定其它针对此环境问题。

当两个或多个运行在同一 Adaptive Server 中的执行对象使用或访问一组公共资源时，它们是侵入性的。

侵入性的应用程序	
指派属性的影响	为侵入性的应用程序指派较高的执行属性可能会使性能下降。
示例	假定这样一个情形：不关键的应用程序已准备好释放资源，但此时有一个非常关键的应用程序开始执行，结果发生阻塞。如果某次关键应用程序需要使用被阻塞的资源，则此次关键应用程序的执行也会被阻塞。

如果 Adaptive Server 环境中的应用程序使用不同的资源，则它们是未入侵的。

未入侵的应用程序	
指派属性的影响	为未入侵的应用程序指派优先的执行属性，可提高性能。
示例	对不同数据库中的表同时进行的 distinct 操作是未入侵的。两个操作中，如果其中一个属于计算密集型而另一个属于 I/O 密集型，则它们也是未入侵的。

I/O 密集型与 CPU 密集型执行对象

当执行对象属于 I/O 密集型时，为它提供 EC1 预定义执行类属性可能有帮助 （请参见第 55 页的 “执行类的类型”）。执行 I/O 的对象通常不使用整个时间段，并且会在等待 I/O 完成之前放弃 CPU。

通过为 I/O 密集型 Adaptive Server 任务提供优先顺序， Adaptive Server 可确保这些任务可在 I/O 结束后立即运行。通过让 I/O 首先执行， CPU 应能够支持 I/O 密集型和计算密集型的应用程序和登录。

第 2 阶段 — 整个环境

作为第 1 阶段 （在该阶段确定了执行对象的行为）的继续，考虑各应用程序如何交互。

通常，一个应用程序在不同时间的表现是不同的；也就是说，它可能间或地是侵入性的或未入侵的，间或地属于或不属于 I/O 密集型和 CPU 密集型。这使得我们难以预测应用程序间的交互作用，但可以判断它们的趋势。

将分析结果组织在一起，以便尽可能地了解每个执行对象与其它对象的关系。例如，可创建一张确定各对象及其行为趋势的表。

使用 Adaptive Server 监控工具 （例如，监控表）是了解执行对象如何影响环境的最佳方式之一。请参见 Performance and Tuning Series: Monitoring Tables （《性能和调优系列：监控表》）。

执行基准测试

在指派任何执行属性前，应执行基准测试，以便在进行调整之后以此测试的结果作为基准。

可帮助您了解系统和应用程序行为的工具包括：

- 监控表 — 提供系统范围的视图或性能以及有关对象和用户的详细信息。请参见 Performance and Tuning Series: Monitoring Tables （《性能和调优系列：监控表》）。

- sp_sysmon — 是一个系统过程，用于监控指定时间间隔内的系统性能，然后输出 ASCII 文本格式的报告。请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

制定目标

建立一组可计量的性能目标。它们应是基于基准结果和对改善性能的期望的具体数字。利用这些目标来指导您指派执行属性。

结果分析与调优

配置执行层次后，分析运行的 Adaptive Server 环境：

1 运行在指派执行属性前运行的同一基准测试，并将结果与基准结果相比较。

2 使用 Adaptive Server Monitor 或 sp_sysmon 确保跨所有可用引擎的合理分配。查看 “内核使用率”。请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

3 如果未达到预期效果，则进一步考查执行对象间的交互作用。查找不合适的假定和可能遗漏的依赖性。

4 调整性能属性。

5 通过按需要多次重复这些步骤来调优结果，以随时间监控环境。

管理对资源的优先访问权

大多数的性能调优技术使您能在系统级或具体的查询级上进行控制。Adaptive Server 也允许对同时运行的多个任务的相关性能进行控制。

除非有过多资源，否则在并行执行环境中更加需要任务级上的控制，因为对有限资源的争用更严重。

使用系统过程来指派执行属性，用以指示哪些任务应拥有对资源的优先访问权。逻辑进程管理器在为任务指派优先级和为引擎指派任务时，会使用执行属性。

实际上，指派执行属性让您通知 Adaptive Server 在混合工作负载环境中如何在客户端应用程序、登录和存储过程之间分配引擎资源。

每个客户端应用程序或登录可启动许多 Adaptive Server 任务。在单一应用程序环境中，可在登录级和任务级分配资源，以提高所选连接或会话的性能。在多应用程序环境中，可为所选应用程序以及所选连接或会话分配资源，以改善它们的性能。

警告！指派执行属性时应谨慎行事。

随意改变一个客户端应用程序、登录或存储过程的执行属性可能会对其它对象的性能产生负面影响。

执行类的类型

执行类是各种执行属性的特定组合，这些执行属性指定任务优先级以及任务 — 线程池关联（或进程模式中的任务 — 引擎密切连接）的值。可将执行类绑定到一个或多个执行对象（客户端应用程序、登录、服务类和存储过程）。

有两种类型的执行类—预定义的和用户定义的。 Adaptive Server 提供三种预定义执行类：

- EC1 — 拥有最优先的属性。
- EC2 — 拥有平均的属性值。
- EC3 — 拥有非优先的属性值。

与 EC2 相关联的对象对引擎资源拥有平均优先级。如果执行对象与 EC1 相关联，则 Adaptive Server 视其为关键对象，并尝试赋予它对资源的优先访问权。

任何与 EC3 相关联的执行对象均被视为最不关键的对象，在与 EC1 和 EC2 相关联的执行对象执行完毕前，此类执行对象不能获得资源。缺省情况下，执行对象拥有 EC2 属性。

若要将执行对象的执行类从缺省的 EC2 更改为其它值，可按第 60 页的“指派执行类”所述使用 sp_bindexeclass。

通过以最能反映网站需要的方式组合执行属性来创建用户定义的执行类。这样做的原因有：

- EC1、 EC2 和 EC3 并没有提供所有可能有用的属性组合。
- 使执行对象与特定的一组引擎相关联可以改善性能。
- 将服务任务 （例如管家任务、 LICENSE HEARTBEAT 等）绑定到各自的线程池

执行类属性

每个预定义或用户定义的执行类包含三种属性的组合：基本优先级、时间片和线程池关联 （或进程模式中的引擎密切连接）。这些属性决定了执行期间的性能特性。

预定义执行类 EC1、 EC2 和 EC3 的属性是固定的 （如表 4-1 所示）。

表 4-1：预定义执行类的固定属性组合

执行类级别	基本优先级 attribute	时间片 attribute	引擎密切连接 attribute
EC1	高	时间片 > t	否
EC2	中	时间片 = t	否
EC3	低	时间片 < t	引擎 ID 号最大的引擎

缺省情况下， Adaptive Server 中的任务以与 EC2 相同的属性进行操作：其基本优先级为中，其时间片设置为一个时钟周期，并可在任意引擎上运行。

基本优先级

在创建任务时指派基本优先级。值为 “high”（高）、“medium”（中）和 “low”（低）。每个引擎的每个优先级有一个运行队列，而全局运行队列也有每个优先级的一个队列。

线程池查找要运行的任务时，首先会检查自己的高优先级运行队列，再检查高优先级的全局运行队列，然后检查自己的中优先级运行队列，依此类推。结果是，高优先级运行队列中的可运行任务比其它队列中的任务更快地调度到线程池。

计划程序搜索空间是指引擎计划程序在哪里查找工作（运行队列检查）：

- 在进程模式中，计划程序搜索空间是服务器范围的，这意味着所有引擎共享一个全局运行队列。引擎检查所有其它引擎的运行队列。
- 在线程化模式中，计划程序搜索空间特定于线程池。每个引擎线程池具有自己的全局队列，并且该池中的引擎只查找与该池关联的任务。

在执行期间， Adaptive Server 可按需要临时更改任务的优先级。任务的优先级可大于或等于其基本优先级，但永远不能小于它。

创建用户定义执行类时，可将值 high（高）、medium（中）或 low（低）指派给任务。

设置任务优先级

任务优先级是使用 sp_bindexeclass 设置的执行类的属性。 sp_showpsexe 输出的 current_priority 列显示当前任务执行设置的优先级：

```
sp_showpsexe
spid           appl_name                        login_name
                     exec_class                       current_priority
           task_affinity
------          ------------------------------ ------------------------------
                    ------------------------------ ----------------
           ------------------------------
     6                                    NULL                                NULL
                                               NULL               LOW
           syb_default_pool
     7                                    NULL                                NULL
                                               NULL            MEDIUM
           syb_default_pool
     8                                    NULL                                NULL
                                               NULL               LOW
           syb_default_pool
    13                                    isql                                  sa
                                                EC2            MEDIUM
            syb_default_pool
```

在线程化模式中， task_affinity 列指示线程池的名称。在进程模式中，它指示引擎组的名称。

使用 sp_setpsexe 可以设置特定任务的优先级。例如，若要将上述示例中的 isql 任务设置为优先级 HIGH，请使用：

```
sp_setpsexe 13, 'priority', 'HIGH'
```

设置任务优先级时，请考虑：

- 为 Adaptive Server 任务而非操作系统线程设置优先级。
- 优先级与其它任务相关。例如，如果用户线程池只包含单个执行类的任务，则设置该类的优先级无效，因为所有任务在同一优先级上运行。

任务 — 引擎密切连接

在多引擎环境中，任何可用引擎都可以处理全局运行队列中的下一个任务。通过引擎密切连接属性可将任务指派给一个引擎或引擎组（在线程化模式中，这通过线程池执行）。

组织任务 — 引擎密切连接：

- 将较不关键的执行对象与已定义的引擎组相关联，以将该对象限制给所有引擎的一个子集。这样会降低这些对象的处理器可用性。较关键的执行对象可在任意 Adaptive Server 引擎上执行，因此其性能得以改善，这是因为可以利用不太关键的执行对象无法使用的资源。
- 将较关键执行对象与已定义的、较不关键对象无权访问的引擎组相关联。这样可确保关键执行对象获得对已知量处理能力的访问权。
- 在进程模式中，当网络密集型任务的最佳性能是主要问题时，管理员可以结合使用任务 — 引擎密切连接和动态监听器来确保任务与所有任务的网络 I/O 运行在同一引擎上。在线程化模式中，由于缺少专用的网络引擎，因此不需要这样做。

EC1 和 EC2 不为执行对象设置引擎密切连接；但 EC3 设置与当前配置中引擎编号最大的 Adaptive Server 引擎的密切连接。

使用 sp_addengine 创建引擎组，并使用 sp_addexeclass 将执行对象绑定到该引擎组。如果不想为用户定义的执行类指派引擎密切连接，请使用 *ANYENGINE* 作为引擎组参数。

在线程化模式中，使用 create thread pool 创建新线程池。使用 sp_addexeclass 将执行对象绑定到线程池。

注释 引擎密切连接属性不用于存储过程。

切换模式时的引擎组密切连接

引擎组不存在于线程化模式中。在从线程化模式切换到进程模式时，会将执行类指派给缺省的引擎组。例如，如果从线程化模式切换到进程模式，然后添加 Eng_Group 执行类并将其与引擎编号 3 相关联，则缺省执行类 EC1 和 EC2 将与 ANYENGINE 引擎组相关联，并且具有最高引擎编号的 EC3 将与 LASTONLINE 引擎组相关联：

```
sp_showexeclass
classname                      priority   engine_group
            引擎
------------------------------ ---------- ------------------------------
              -------------------------------
EC1                                  HIGH                        ANYENGINE
                                        ALL
EC2                                MEDIUM                        ANYENGINE
                                        ALL
EC3                                   LOW                       LASTONLINE
                                          0
Eng_Group                             LOW                 new_engine_group
                                          3
```

在切换到线程化模式时，执行类会丢失其引擎组密切连接并将指派给 syb_default_pool。在线程化模式中，上述示例变成：

```
sp_showexeclass
classname                     priority     threadpool
----------------------------- ----------   ------------------------------
EC1                           HIGH         syb_default_pool
EC2                           MEDIUM       syb_default_pool
EC3                           LOW          syb_default_pool
Eng_Group                     LOW          new_engine_group
```

设置执行类属性

使用表 4-2 中列出的系统过程类别，实施和管理客户端应用程序、登录、服务任务和存储过程的执行层次。

表 4-2：管理执行对象优先顺序的系统过程

类别	说明 (Description)	系统过程
用户定义执行类	创建或删除带自定义属性的用户定义类，或更改现有类的属性。	• sp_addexeclass • sp_dropexeclass
执行类绑定	将预定义类或用户定义类绑定到客户端应用程序、服务任务和登录，或解除绑定。	• sp_bindexeclass • sp_unbindexeclass
仅用于当前会话（“随即”）	仅设置和清除活动会话的属性。	• sp_setpsexe • sp_clearpsexe
引擎	将引擎添加到引擎组中，或从引擎组中删除引擎；创建和删除引擎组。	• sp_addengine • sp_dropengine
报告	报告引擎组指派、应用程序绑定和执行类属性。	• sp_showcontrolinfo • sp_showexeclass • sp_showpsexe

请参见 《参考手册：过程》。

指派执行类

下例说明如何通过将执行对象与 EC1 执行类相关联，来为其指派对资源的优先访问权。在这种情况下，执行对象是应用程序和登录的组合。

例如，如果决定 “sa” 登录必须尽快从 isql 获得结果，请使用优先执行类 EC1 发出 sp_bindexeclass，以便 Adaptive Server 在执行 isql 时，授予登录 “sa” 执行优先顺序：

```
sp_bindexeclass sa, LG, isql, EC1
```

此语句指定当名为 “sa” 的登录 (LG) 执行 isql 应用程序时，“sa” 登录任务使用 EC1 属性执行。Adaptive Server 通过以下方法缩短 “sa” 登录的响应时间：将它放于高优先级运行队列中，以便更快地将它指派给引擎。

调度服务任务

Adaptive Server 允许使用 sp_bindexeclass 'sv' 执行类参数管理服务任务（管家、检查点、 Replication Agent 线程等）。将各个服务任务绑定到执行类会将这些任务绑定到线程池，这允许您控制高优先级任务的专用资源，并防止服务任务与用户任务竞争。

注释　在进程模式中，无法调度服务任务。

例如，可以：

- 将 HK WASH 管家任务绑定到特定服务任务。
- 对每个 Replication Agent 使用一个线程来建立 Replication Agent 池和执行类，从而提供专用资源，但同时创建名为 service_pool 的更加通用的线程池，将一个线程授予不太重要的其它任务。

monServiceTask 监控表包括有关绑定到执行类的所有服务任务的信息。此示例显示绑定到 SC 执行类的 HK WASH 和 NETWORK HANDLER 服务任务：

```
task_id    spid        name
description                 execution_class
---------- ----------- ----------------------------
------------------------------ ------------------------
3932190               6                          HK WASH
                          NULL                         SC
4456482              10                 NETWORK HANDLER
                          NULL                         SC
```

创建用户定义执行类任务关联

以下步骤说明如何使用系统过程创建与用户定义执行类相关联的线程池，以及如何将该执行类绑定到用户会话。在此例中，服务器由必须尽快响应客户需要的技术支持人员和通常负责编译报告并能承受较慢响应时间的管理人员使用。

为此示例创建用户定义执行类：

1　创建名为 DS_GROUP 的线程池来控制任务。例如：

```
create thread pool DS_GROUP with thread count = 4
```

2　使用 sp_addexeclass 创建具有您选择的名称和属性的用户定义执行类。例如：

```
sp_addexeclass DS, LOW, 0, DS_GROUP
```

3 使用 sp_bindexeclass 将用户定义执行类与执行对象相关联。例如使用三个登录：

```
sp_bindexeclass mgr1, LG, NULL, DS
sp_bindexeclass mgr2, LG, NULL, DS
sp_bindexeclass mgr3, LG, NULL, DS
```

如果 Adaptive Server 配置为进程模式，请执行以下步骤创建用户定义执行类：

1 创建包含引擎 3、名为 DS_GROUP 的引擎组：

```
sp_addengine 3, DS_GROUP
```

扩展组，以便它还包括引擎 4 和 5：

```
sp_addengine 4, DS_GROUP
sp_addengine 5, DS_GROUP
```

2 创建名为 DS、优先级为 “low” 的用户定义执行类并将其与 DS_GROUP 引擎组相关联。

```
sp_addexeclass DS, LOW, 0, DS_GROUP
```

3 将不太关键的执行对象绑定到新执行类。

例如，三次使用 sp_bindexeclass 将管理人员登录 “mgr1”、“mgr2” 和 “mgr3” 绑定到 DS 执行类：

```
sp_bindexeclass mgr1, LG, NULL, DS
sp_bindexeclass mgr2, LG, NULL, DS
sp_bindexeclass mgr3, LG, NULL, DS
```

第二个参数 LG 指示第一个参数是登录名。第三个参数 NULL 表示这种关联应用于该登录可能运行的任意应用程序。第四个参数 DS 表示登录被绑定到 DS 执行类。

此例的结果是赋予技术支持组 （未绑定到引擎组）比管理人员更快的、对处理资源的访问权。

执行类绑定如何影响调度

可采用逻辑进程管理来增加特定登录、特定应用程序或执行特定应用程序的特定登录的优先级。此例针对：

- order_entry 应用程序，它是对获得客户订单非常关键的一个 OLTP 应用程序。
- sales_report 应用程序，它准备各种报告。一些管理人员用缺省特性运行此应用程序，而其它管理人员则以较低的优先级运行报告。
- 其他用户，他们以缺省优先级运行其它应用程序。

执行类绑定

以下语句将 order_entry 与 EC1 属性绑定，为运行它的任务赋予较高的优先级：

```
sp_bindexeclass order_entry, AP, NULL, EC1
```

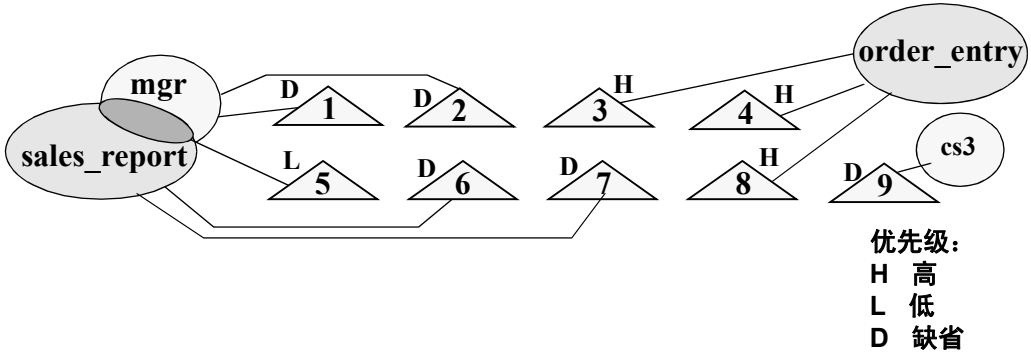
以下 sp_bindexeclass 语句指定 “mgr”以 EC3 运行 sales_report 应用程序：

```
sp_bindexeclass mgr, LG, sales_report, EC3
```

此任务只能在没有具有 EC1 或 EC2 属性的可运行任务时执行。

图 4-2 显示了四个运行任务的执行对象。多个用户正在运行 order_entry 和 sales_report 应用程序。其它两个登录处于活动状态，一个为 “mgr”（一次使用 sales_report 应用程序登录，两次使用 isql 登录），一个为 “cs3”（未使用受影响的应用程序）。

图 4-2：执行对象及其任务



当 “mgr”登录使用 isql 时 （任务 1 和 2），任务以缺省属性运行。而当“mgr”登录使用 sales_report 时，任务以 EC3 运行。其他运行 sales_report 的管理人员（任务 6 和 7）以缺省属性运行。所有运行 order_entry 的任务以高优先级、EC1 属性运行（任务 3、4 和 8）。“cs3”以缺省属性运行。

引擎密切连接可影响进程模式中的调度

引擎在查找要运行的任务时，首先在自己的高优先级运行队列中查找，然后在高优先级全局运行队列中查找。如果没有高优先级任务，则引擎先在自己的运行队列，然后在中优先级全局运行队列中查找中优先级任务，最后再查找低优先级任务。

如果任务与特定引擎有密切连接会怎样？假定图 4-2 中的任务 7 是全局运行队列中的一个高优先级任务，它拥有一个高优先级的、与引擎编号 2 密切连接的用户定义执行类，但此引擎的队列中目前排列有几个高优先级任务，而它正在运行另一个任务。

如果当引擎 1 处理完图 4-2 中的任务 8 时，其队列中没有高优先级任务，它便会查看全局运行队列，但由于存在引擎绑定，它不能处理任务 7。然后引擎 1 会查看自己的中优先级队列，运行任务 15。尽管系统管理员指派了优先执行类 EC1，但引擎密切连接临时使任务 7 的执行优先顺序降低到执行类为 EC2 的任务之下。

这种影响可能不理想，也可能是所期望的。您可指派引擎密切连接和执行类，以便任务优先级不是自己所期望的。也可以进行指派，以便低优先级任务可能永不运行或等待相当长的时间 — 这是指派执行类和引擎密切连接时需要进行全面计划和测试的重要原因。

注释 在线程化模式中，引擎及其指派的任务存在于完全独立的搜索空间中。

仅为会话设置属性

使用 sp_setpsexe 可以临时更改活动会话的任何属性值。

此属性更改仅对指定的 spid 并仅在此会话期间有效，无论此会话是自然结束还是被终止。使用 sp_setpsexe 设置属性既不会改变该执行类关于其它任何进程的定义，也不会应用到对其使用此命令的活动进程的下一次调用。

若要清除会话的属性设置，请使用 sp_clearpsexe。

获取有关执行类的信息

Adaptive Server 将有关执行类指派的信息存储在系统表 sysattributes 和 sysprocesses 中，并支持用于确定已进行了哪些指派的多种系统过程。

使用 sp_showcontrolinfo 显示绑定到执行类的执行对象、引擎组中的 Adaptive Server 引擎及会话级属性绑定的有关信息。如果不指定参数，sp_showcontrolinfo 会显示整个绑定集和所有引擎组的组成部分。

sp_showexeclass 显示某执行类或所有执行类的属性值。

也可以使用 sp_showpsexe 来查看所有运行进程的属性。

确定优先顺序与范围

确定两个或多个执行对象间的最终执行层次可能会非常复杂。当相关执行对象与各种不同执行属性的组合使得执行顺序混乱不清时，情况会怎样呢？

例如，EC3 客户端应用程序可调用 EC1 存储过程。两个执行对象都具有 EC3 属性、 EC1 属性，还是 EC2 属性？

了解 Adaptive Server 如何确定执行优先顺序，对于通过执行类指派获得预期效果是十分重要的。优先规则和范围规则这两个基本规则可帮助您确定执行顺序。

多个执行对象和 EC

Adaptive Server 采用优先规则和范围规则来确定应用多个冲突规范中的哪一个。

使用规则时应遵循以下顺序：

1　当进程涉及多个执行对象类型时使用优先规则。

2　当同一个执行对象存在多个执行类定义时使用范围规则。

优先规则

当属于一个执行类的执行对象调用属于另一执行类的执行对象时，优先规则会排出执行优先顺序。

优先规则规定存储过程的执行类优先于登录的执行类，而登录的执行类优先于客户端应用程序的执行类。

如果存储过程的某执行类优先于调用它的客户端应用进程的执行类，则在存储过程运行期间，客户端进程的优先级临时被提高到此存储过程的优先级。这也适用于嵌套的存储过程。

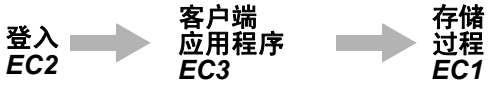
注释 优先规则例外：如果某执行对象调用一个执行类优先级低于自己的存储过程，则该执行对象的优先级不会临时降低。

优先规则示例

本例说明优先规则的使用。假定有一个 EC2 登录、一个 EC3 客户端应用程序和一个 EC1 存储过程。

登录的属性优先于客户端应用程序的属性，因此登录被优先处理。如果存储过程的基本优先级高于登录的优先级，则在存储过程的执行期间，执行存储过程的 Adaptive Server 进程的基本优先级临时提高。图 4-3 说明了如何应用优先规则。

图 4-3：优先规则的使用



存储过程以 *EC1* 运行

当具有 EC2 的登录调用具有 EC1 的客户端应用程序，而此客户端应用程序调用具有 EC3 的存储过程时，情况会怎样呢？由于登录的执行类优先于客户端应用程序的执行类，因此存储过程会以 EC2 属性执行。使用上面注释中说明的优先规则例外，优先级不会临时降低。

范围规则

除指定对象的执行属性外，还可以在使用 sp_bindexeclass *scope* 时定义其范围。对象的范围指定执行类绑定对其有效的实体

例如，可指定 isql 客户端应用程序以 EC1 属性运行，但必须是在它以“sa”登录执行时。此语句将绑定到 isql 客户端应用程序的 EC1 的范围设置为 “sa”登录 （AP 指示应用程序）：

```
sp_bindexeclass isql, AP, sa, EC1
```

反之，也可指定 “sa”登录以 EC1 属性运行，但必须是在它执行 isql 客户端应用程序时。在此示例中，绑定到 “sa”登录的 EC1 的范围为 isql 客户端应用程序：

```
sp_bindexeclass sa, LG, isql, EC1
```

如果范围设置为 NULL，则绑定适用于所有交互。

当客户端应用程序没有范围时，绑定到它的执行属性应用于任何调用该客户端应用程序的登录。

登录没有范围时，对于登录的任何进程，该属性都适用。

以下命令中的 isql 参数指定，对于调用 isql 的任何登录， Transact-SQL 应用程序使用 EC3 属性执行，除非该登录绑定到较高的执行类：

```
sp_bindexeclass isql, AP, NULL, EC3
```

结合以上授予 “sa”用户 isql EC1 执行属性的绑定，并使用优先规则，来自 “sa”登录的 isql 请求使用 EC1 属性执行。为来自非 “sa”登录的 isql 请求服务的其它进程使用 EC3 属性执行。

范围规则规定，当为客户端应用程序、登录、服务类或存储过程指派了多个执行类级别时，范围最窄的级别优先。利用范围规则，使用以下命令可获得同样的结果：

```
sp_bindexeclass isql, AP, sa, EC1
```

解决优先冲突

当多个执行对象和执行类拥有相同的范围时， Adaptive Server 使用以下规则解决优先冲突。

- 给未绑定到特定执行类的执行对象指派这些缺省值：

实体类型	属性名	缺省值
客户端应用程序	执行类	EC2
登入	执行类	EC2
存储过程	执行类	EC2

- 指派了执行类的执行对象的优先级高于缺省值。（已指派的 EC3 优先于未指派的 EC2）。
- 如果客户端应用程序和登录有不同的执行类，则登录的执行优先级高于客户端应用程序的优先级 （依据优先规则）。
- 如果存储过程和客户端应用程序或登录有不同的执行类， Adaptive Server 执行存储过程时，会使用有较高执行类的那个，以产生优先顺序 （依据优先规则）。
- 如果同一个执行对象有多个定义，则范围最窄的定义具最高的优先级 （依据范围规则）。例如，第一个语句赋予运行 isql 的 “sa” 登录比运行任何其它任务的 “sa” 登录更高的优先级：

```
sp_bindexeclass sa, LG, isql, EC1
sp_bindexeclass sa, LG, NULL, EC2
```

示例：确定优先顺序

表 4-3 中的每一行包含执行对象及其冲突执行属性的组合。

“执行类属性” 列显示指派给属于登录 “LG” 的进程应用程序 “AP” 的执行类值。

其它列显示 Adaptive Server 如何解析优先顺序。

表 4-3：冲突属性值与 Adaptive Server 指派值

执行类属性			Adaptive Server 指派值		
应用程序 (AP)	登录 (LG)	存储过程 (sp_ec)	应用程序	登录基本优先级	存储过程基本优先级
EC1	EC2	EC1 (EC3)	EC2	中	高（中）
EC1	EC3	EC1 (EC2)	EC3	低	高（中）
EC2	EC1	EC2 (EC3)	EC1	高	高（高）
EC2	EC3	EC1 (EC2)	EC3	低	高（中）
EC3	EC1	EC2 (EC3)	EC1	高	高（高）
EC3	EC2	EC1 (EC3)	EC2	中	高（中）

若要测试您对优先规则和范围规则的理解，请遮住表 4-3 的“Adaptive Server 指派值”列，并预测这些列中的值。为帮助您入门，以下是对第一行的情况说明：

- 列 1 — 将客户端应用程序 AP 指定为 EC1。
- 列 2 — 将登录“LG”指定为 EC2?
- 列 3 — 存储过程 sp_ec 被指定为 EC1?

运行时：

- 列 4 — 属于 LG 的执行客户端应用程序 AP 的任务使用 EC2 属性，因为登录的类优先于应用程序的类（优先规则）。
- 列 5 — 列 5 的值表示此登录的基本优先级为中。
- 列 6 — 存储过程 sp_ec 的执行优先级从中提升至高（因为它是 EC1）。

  如果将存储过程指派为 EC3（如列 3 圆括号中所示），则存储过程的执行优先级为中（如列 6 圆括号中所示），因为 Adaptive Server 使用客户端应用程序或登录和存储过程的最高执行优先级。

使用优先规则的示例

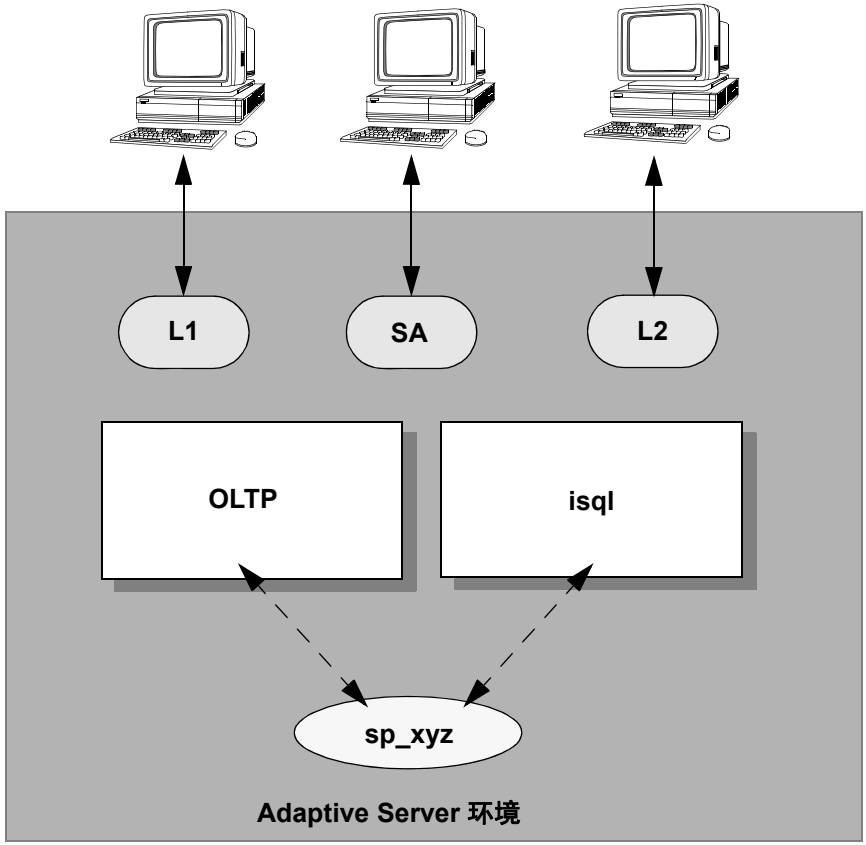
本节举例说明系统管理员如何解释执行类属性，其中包括：

- 规划 — 系统管理员分析环境、执行基准测试、设置目标和了解足以预测结果的概念。
- 配置 — 系统管理员使用基于 “规划”部分中收集的信息的参数运行 sp_bindexeclass。
- 执行特征 — 应用程序使用系统管理员创建的配置与 Adaptive Server 连接。

图 4-4 显示了两个客户端应用程序 OLTP 和 isql，及三个 Adaptive Server 登录 “L1”、“sa” 和 “L2”。

sp_xyz 是 OLTP 应用程序和 isql 应用程序需要执行的存储过程。

图 4-4：冲突解决方案



计划

系统管理员按第 49 页的 “成功分配资源”中的第 1 步和第 2 步所述进行分析，并定出以下层次计划：

- OLTP 应用程序是一个 EC1 应用程序，而 isql 应用程序是一个 EC3 应用程序。
- 登录 “L1”可在不同时间运行不同的客户端应用程序，并且没有特殊的性能要求。
- 登录 “L2”是一个较不关键的用户，应始终以低性能特性运行。
- 登录 “sa”必须始终作为关键用户运行。
- 存储过程 sp_xyz 应始终以高性能特性运行。由于 isql 客户端应用程序可执行存储过程，因此赋予 sp_xyz 高性能特性可能会避免在 OLTP 客户端应用程序的路径上产生瓶颈。

表 4-1 总结了分析，并指定了系统管理员要指派的执行类。如将此表降格，则粒度调优将更精细。应用程序的粒度最大，或者说范围最广。存储过程的粒度最佳，或者说范围最窄。

表 4-4：Adaptive Server 环境分析示例

标识符	交互作用与注释	执行类
OLTP	• 与 isql 有相同的表 • 非常关键	EC1
isql	• 与 OLTP 有相同的表 • 低优先级	EC3
L1	• 未指派优先级	否
sa	• 非常关键	EC1
L2	• 不关键	EC3
sp_xyz	• 避免产生 “热点”	EC1

配置

系统管理员执行以下系统过程来指派执行类 （第 56 页上的第 3 步）：

```
sp_bindexeclass OLTP, AP, NULL, EC1
sp_bindexeclass ISQL, AP, NULL, EC3
sp_bindexeclass sa, LG, NULL, EC1
sp_bindexeclass L2, LG, NULL, EC3
sp_bindexeclass SP_XYZ, PR, sp_owner, EC1
```

执行特性

以下是在本例描述的配置下、在 Adaptive Server 环境中可能发生的一系列事件：

1 客户端使用 OLTP 以 “L1” 身份登录到 Adaptive Server。

- Adaptive Server 确定 OLTP 为 EC1。
- “L1” 没有执行类。不过，因为 “L1” 登录到 OLTP 应用程序，Adaptive Server 指派执行类 EC1。
- “L1” 以高优先级执行存储过程，因为为对象指派了执行类 EC1。

2 客户端使用 isql 以 “L1” 身份登录到 Adaptive Server。

- 因为 isql 为 EC3，而 “L1” 未绑定到执行类，所以 “L1” 以 EC3 特性执行。也就是说，它以低优先级运行，并与最大编号的引擎密切连接 （只要有多个引擎）。
- 当 “L1” 执行 sp_xyz 时，因为存储过程为 EC1，所以 “L1” 的优先级提升为高。

3 客户端使用 isql 以 “sa” 身份登录到 Adaptive Server。

- Adaptive Server 使用优先规则确定 isql 和 “sa” 的执行类。Adaptive Server 使用 EC1 属性运行系统管理员的 isql 实例。当系统管理员执行 sp_xyz 时，优先级不变。

4 客户端使用 isql 以 “L2” 身份登录到 Adaptive Server。

- 由于应用程序和登录同为 EC3，因此不存在冲突。“L2” 以高优先级执行 sp_xyz。

引擎资源分配的注意事项

不加区分地指派执行类通常不会获得预期的效果。对每种类型的执行对象而言，某些条件会产生更佳的性能。表 4-5 指出对每种执行对象来说，何时指派执行优先顺序会更有利。

表 4-5：何时指派执行优先顺序有益

执行对象	说明
客户端应用程序	各客户端应用程序间对非 CPU 资源几乎不存在争用现象。
Adaptive Server login	一个登录对 CPU 资源的优先级应高于其它登录。
存储过程	存在预定义的存储过程 “热点”。

降低不太关键执行对象的执行类比提高非常关键执行对象的执行类更有效。

客户端应用程序：OLTP 和 DSS

当客户端应用程序之间几乎不存在对非 CPU 资源的争用情况时，为客户端应用程序指派更高的执行优先级会更有益。

例如，一个 OLTP 应用程序和一个 DSS 应用程序并发执行时，您也许会愿意牺牲 DSS 应用程序的性能，如果这样做能够使 OLTP 应用程序更快地执行。那么，可为 DSS 应用程序指派非优先的执行属性，使它在 OLTP 任务执行完毕后才能获得 CPU 时间。

未入侵的客户端应用程序

对于使用或访问未被系统中任何其它应用程序使用的表的未入侵的应用程序来说，应用程序间锁争用不是问题。

为此类应用程序指派优先执行类可确保当此应用程序有可运行的任务时，它会排在等待 CPU 时间队列的首位。

I/O 密集型客户端应用程序

如果非常关键的应用程序属于 I/O 密集型，而其它应用程序属于计算密集型，则在计算密集型进程未因为某个其它原因被阻塞的情况下，它使用 CPU 的时间可达整个时间片。

不过， I/O 密集型进程会在每次执行 I/O 操作时放弃 CPU。为计算密集型应用程序指派非优先执行类后， Adaptive Server 可更快地运行 I/O 密集型进程。

关键应用程序

如果在多个非关键执行对象中存在一两个关键执行对象，则应尝试为不太关键的应用程序设置与特定线程池的关联。这样可提高关键应用程序的吞吐量。

Adaptive Server 登录：高优先级用户

如果为关键用户指派了优先执行属性，而使其他用户保持缺省属性，则 Adaptive Server 会尽可能地首先执行所有与高优先级用户相关联的任务。

在进程模式中，调度的一个结果是当引擎在其本地运行或全局运行队列中找不到任务时，会尝试从其它引擎的本地运行队列中挪用任务。引擎只能挪用具有普通优先级的任务，而决不能挪用高优先级用户的高优先级任务。如果引擎负载的平衡状态不好，并且运行高优先级任务的引擎的负载很重，则任务挪用可导致高优先级任务缺乏 CPU，这与调度的预期影响相反，是固有的副作用。

存储过程：“热点”

当存储过程被一个或多个应用程序过度使用时，便会出现与存储过程相关的性能问题。出现此类问题时，存储过程在应用程序的路径中表现出热点特性。

通常，执行存储过程的应用程序的执行优先级在中到低的范围内，因此为存储过程指派更优先的执行属性可能会改善调用它的应用程序的性能。

第 5 章

内存使用和性能

本章介绍 Adaptive Server 如何使用数据和过程高速缓存，以及其它受内存配置影响的问题。通常，可用内存越多， Adaptive Server 的响应速度越快。

《系统管理指南，卷 2》中的第 3 章 “配置内存”介绍了如何确定 Adaptive Server 的最佳内存配置值，以及其它服务器配置选项需要的内存。

内存如何影响性能

充足的内存可减少磁盘 I/O，因为内存访问比磁盘访问快很多，所以提高了性能。用户发出查询时，数据和索引页必须在内存中或已读入内存中，才能检查其中的值。如果这些页已驻留在内存中，Adaptive Server 不需要执行磁盘 I/O。

添加更多内存廉价而便捷，但围绕内存问题的开发却是昂贵的。尽可能为 Adaptive Server 提供更多的内存。

可导致性能低下的内存条件包括：

- 总数据高速缓存太小。
- 过程高速缓存太小。
- 有几个活动 CPU 的 SMP 系统上只配置了缺省的高速缓存，导致对数据高速缓存的争用。
- 用户配置的数据高速缓存大小不适合特定的用户应用程序。
- 配置的 I/O 大小不适合特定的查询。
- 审计队列大小不合适 （如果安装了审计功能）。

要配置多少内存

配置 Adaptive Server 时，内存是最重要的考虑因素。内存由各种配置参数、线程池、过程高速缓存和数据高速缓存使用。正确设置配置参数和高速缓存的值是获得良好系统性能的关键。

启动过程中分配的内存总量是为满足所有 Adaptive Server 配置要求所需的内存总和。此值由 Adaptive Server 通过只读配置参数 total logical memory 进行累积。配置参数 max memory 必须大于或等于 total logical memory。 max memory 表示用于满足 Adaptive Server 需要的内存量。

Adaptive Server 在启动时会根据 total logical memory 的值分配内存。不过，如果已经设置了配置参数 allocate max shared memory，则 Adaptive Server 分配的内存量将基于 max memory 的值。这将允许系统管理员告知 Adaptive Server 在启动时分配允许的最大值，而此时该值可能明显大于 total logical memory 的值。

内存配置的要点是：

- 系统管理员应确定 Adaptive Server 可用的共享内存的大小，并将 max memory 设置为此值。
- 将 allocate max shared memory at startup 配置参数的值设置为最少量的共享内存段。这可以改善性能，因为在某些平台上，使用大量共享内存段可能导致性能下降。请查看操作系统文档，以确定最佳共享内存段数量。分配共享内存段后，在下次启动 Adaptive Server 之前，将无法释放该共享内存段。
- 可用于新线程池的内存量取决于 max memory 中的可用内存量。如果 Adaptive Server 没有足够的内存来创建线程池，它将显示一条错误消息，指示必须增加多少最大内存量才能创建该线程池。在此示例中

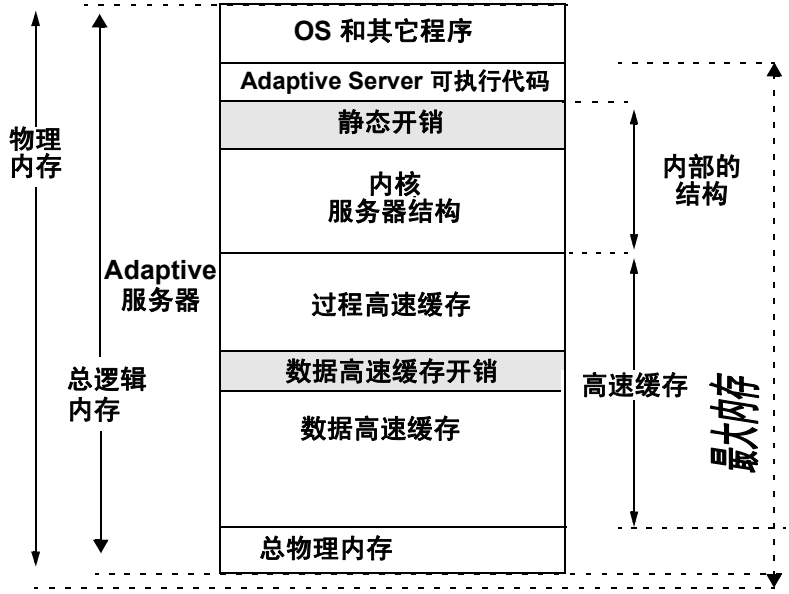
- 如果缺省内存量不够，请重新配置配置参数。
- max memory 与 total logical memory 之间的差值是可供过程、数据高速缓存、线程池或其它配置参数使用的额外内存。

  启动过程中 Adaptive Server 要分配的内存量由 total logical memory 或 max memory 决定。如果此值过高：

  - 在计算机上的物理资源不足的情况下， Adaptive Server 可能无法启动。
  - 如果 Adaptive Server 能够启动，则操作系统页面错误率可能会显著提高，您可能需要重新配置操作系统以降低错误率。

满足所有其它内存要求后剩余的内存用于过程高速缓存和数据高速缓存。图 5-1 显示了如何划分内存。

图 5-1：Adaptive Server 如何使用内存



动态重新配置

Adaptive Server 允许用户动态分配 total physical memory。许多消耗内存的配置参数都是动态的，这意味着无需重新启动服务器便可使其生效。例如，可以对 number of user connections、 number of worker processes 和 time slice 进行动态更改。请参见 《系统管理指南，卷 1》中的第 5 章 “设置配置参数”，以查看配置参数的完整论述，其中包括有关哪些参数是动态的、哪些是静态的信息。

内存分配方式

在 Adaptive Server 12.5 之前的版本中，过程高速缓存的大小基于可用内存的百分比。在用户配置了数据高速缓存后，所剩下的就被分配给过程高速缓存。在 Adaptive Server 12.5 及更高版本中，都按绝对值指定数据高速缓存和过程高速缓存。在用户重新配置前，高速缓存的大小始终保持不变。

使用配置参数 max memory 可设定最大设置，超出该设置后您将不能配置 Adaptive Server 的总物理内存。

Adaptive Server 中的大分配

Adaptive Server 可自动调优过程高速缓存分配的大小，以优化内存使用和减少外部分段。在满足重复的内部内存请求时， Adaptive Server 最初分配 2K 块，然后根据过去的分配历史记录，将分配大小扩展到最大值 16K 块。除了有助于改善性能外，此优化还对最终用户透明。

中的高速缓存Adaptive Server

Adaptive Server 包括过程和数据高速缓存：

- **过程高速缓存**用于存储过程和触发器以及满足短期内存需求，如并行查询的统计信息和查询计划。

  可使用 sp_configure, “procedure cache size” 将过程高速缓存大小设置为一个绝对值。请参见 《系统管理指南，卷 1》中的第 5 章 “设置配置参数”。

- **数据高速缓存**用于数据、索引和日志页。数据高速缓存可划分为单独的命名高速缓存，其中特定数据库或数据库对象绑定到特定的高速缓存。

配置过程高速缓存和数据高速缓存后，即无法再对剩余内存进行划分。

高速缓存大小和缓冲池

Adaptive Server 对高速缓存和缓冲池使用不同的页大小：

- 内存页 — （最大内存、总逻辑内存等）是 2K 的倍数
- 过程高速缓存 — 按 2K 页进行配置
- 缓冲区高速缓存 — 以逻辑页大小单位表示
- 大 I/O — 根据扩充进行度量 （每个扩充是 8 页）。例如，如果 Adaptive Server 配置为 8K 逻辑页大小，则大 I/O 使用的读取或写入为 64K。

如果您启动 Adaptive Server，并且高速缓存是使用对当前逻辑页大小无效的缓冲池定义的，那么当在每个命名高速缓存中为缺省缓冲池配置高速缓存时，将重新分配为这种不适当的缓冲池提供的所有内存。

在如何设置逻辑页大小和缓冲池大小方面，应小心谨慎。

逻辑页大小	可能的缓冲池大小
2K	2K、 4K、 16K
4K	4K、 8K、 16K、 32K
8K	8K、 16K、 32K、 64K
16K	16K、 32K、 64K、 128K

过程高速缓存

Adaptive Server 维护存储过程查询计划的 MRU/LRU （最近使用最多的/最近使用最少的）链。用户执行存储过程时， Adaptive Server 在过程高速缓存中查找要使用的查询计划。如果某一查询计划可用，它就被放置在链的 MRU 端，然后开始执行。

如果内存中无计划，或所有计划都在使用中，则从 sysprocedures 表中读取此过程的查询树。然后使用提供给该过程的参数优化查询树，并将其放在链的 MRU 端，再开始执行。在页链的 LRU 端未用的计划在高速缓存中老化。

为过程高速缓存分配的内存拥有所有批处理 （包括所有触发器）的已优化的查询计划 （偶尔是树）。

如果多个用户同时使用一个过程或触发器，那么在高速缓存中将有它的多个副本。如果过程高速缓存太小，用户尝试执行存储过程或引发触发器的查询时，会收到一条错误消息，且必须重新提交查询。未使用的计划在高速缓存中老化后，所占用的空间即变为可用。

Adaptive Server 在启动时使用缺省过程高速缓存大小 （以内存页为单位）。过程高速缓存的最优值因应用程序的不同而不同，也可因使用模式的变化而不同。使用 procedure cache size 可确定过程高速缓存的当前大小（请参见《系统管理指南，卷 1》中的第 5 章“设置配置参数”）。

获得有关过程高速缓存大小的信息

启动 Adaptive Server 时，错误日志说明有多少过程高速缓存可以使用。

- proc buffers 表示可同时驻留在过程高速缓存中的已编译过程对象的最大数量。
- proc headers 表示过程高速缓存专用的页数。高速缓存中的每个对象都至少需要一页。

监控过程高速缓存性能

sp_sysmon 报告存储过程的执行情况和必须从磁盘中读取存储过程的次数。

请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

如果没有足够的内存装载另一查询树或计划，或者已使用最大数量的编译对象， Adaptive Server 将报告 Error 701。

过程高速缓存大小确定

在生产服务器上，应尽可能减少从磁盘进行的过程读取数。当用户需要执行某过程时， Adaptive Server 应能够在过程高速缓存中为最常用的过程找到一个未用的树或计划。服务器在高速缓存中找到可用计划的百分比称为**高速缓存命中率**。保持高速缓存中过程的较高高速缓存命中率可提高性能。

图 5-2 中的公式给出了一个好的起点。

图 5-2：确定过程高速缓存大小的公式

过程高速缓存大小 = （最大并发用户数） * （4 + 最大计划大小） * 1.25

所需的最小过程高速缓存大小 = （主要过程数） * （平均计划大小）

如果已嵌套存储过程 （过程 A 调用过程 B，后者调用过程 C），则这些存储过程必须同时位于高速缓存中。将嵌套过程的大小相加，用最大总和代替图 5-2 中的公式中的 “最大计划大小”。

最小过程高速缓存大小是允许至少每一常用编译对象的一个副本驻留在高速缓存中的最小内存量。不过，过程高速缓存还可以在执行时作为附加内存用于排序和查询优化以及其它目的。另外，所需内存基于查询的类型。

使用 of sp_monitorconfig 可配置过程高速缓存：

1　将过程高速缓存配置为上面确定的最小大小。

2　运行常规数据库装载。如果出现错误 701，请增加过程高速缓存大小。调整增加的大小，以避免过量分配。建议的增加量是 （128 * （过程高速缓存的大小，以 GB 计））。如果过程高速缓存大小小于 1GB，则按 128MB 的增量增加大小。如果过程高速缓存大小大于 1GB 但小于 2GB，则按 256MB 的增量增加大小，依此类推。

3　当 Adaptive Server 达到或超过高峰装载量后，运行 sp_monitorconfig “procedure cache size”。

4　如果 sp_monitorconfig 指示 Max_Used 远远低于 sp_configure 中过程高速缓存的当前值，表明过量分配了过程高速缓存。应考虑降低 procedure cache size 配置值，以便在下次重新启动时可以分配少量的过程高速缓存。

5 如果 sp_monitorconfig 的 Num_Reuse 输出值是零以外的其它值，则也表明过程高速缓存不足。如果该值在一段时间内增大了，请考虑按照上述步骤 2 中的建议增加过程高速缓存的大小。

估计存储过程大小

sysprocedures 存储过程的规范化查询数。涉及其它开销，可以按每 2K 页两行来考虑此大小。若要估计单个存储过程、视图或触发器的大小，请使用：

```
select count(*) as "approximate size in KB"
from sysprocedures
where id = object_id("procedure_name")
```

例如，要查找 pubs2 中 titleid_proc 的大小，可使用以下命令：

```
select count(*)
from sysprocedures
where id = object_id("titleid_proc")

approximate size in KB
----------------------
                     3
```

如果计划位于高速缓存中，则 monCachedProcedures 监控表包括该计划的大小。

估计用于排序的过程高速缓存大小

若要确定用于排序 （用于 create index、 update statistics、 order by、 distinct、 sort 和 merge join）的过程高速缓存的大小，应首先确定每页的行数：

$$\text{每页行数} = \frac{\text{页的大小}}{\text{行的最小长度}}$$

使用以下公式可确定用于排序的过程高速缓存的大小：

过程高速缓存大小 = （排序缓冲区数） x （每页行数） x 85 字节

注释　如果使用 64 位系统，则此公式使用 100 字节。

估计由 create index 使用的过程高速缓存量

create index 在索引键内排序数据。此排序操作可能需要进行一个或多个内存中排序，以及一个或多个基于磁盘的排序。 Adaptive Server 将 create index 数据排序装载到与正在其上创建索引的表关联的数据高速缓存中。此排序操作使用的数据高速缓存缓冲区的数目受 number of sort buffers 的值的限制。如果要排序的所有键与 number of sort buffers 的值匹配， Adaptive Server 将执行单个内存中排序操作。如果排序缓冲区无法容纳要排序的所有键，则 Adaptive Server 必须将内存中排序的结果写入磁盘，以便 Adaptive Server 可以将下一组索引键装载到要排序的排序缓冲区中。

除了从数据高速缓存分配的排序缓冲区外，此排序操作还需要过程高速缓存中大约 66 字节的元数据。以下是用于计算所使用的过程高速缓存量的公式 （假定使用所有排序缓冲区）：

$$\text{所需 2K 过程高速缓存缓冲区数} = \frac{(\textit{每页行数}) \text{ X } (\textit{排序缓冲区数}) \text{ X 66 字节}}{2048\text{（假定 2K 页大小）}}$$

示例 1

在此示例中，

- number of sort buffers 设置为 500
- create index 创建一个 15 字节字段，因此每页 131 行

因此，将使用所有 500 个 2K 缓冲区容纳要排序的数据，过程高速缓存使用 2,111 个 2K 缓冲区：

(131 X 500 X66) / 2048 = 2,111 个 2K 缓冲区

示例 2

在此示例中，

- number of sort buffers 设置为 5000
- create index 创建一个 8 字节字段，因此每页大约 246 行

因此，将使用所有 5,000 个 2K 缓冲区容纳要排序的数据，过程高速缓存使用 39, 639 个 2K 缓冲区：

(246 X 5,000 X 66) / 2048 = 39,639 个 2K 缓冲区

示例 3

在此示例中：

- number of sort buffers 设置为 1000
- 表较小，您可以完全将其装载到使用 800 个 2K 排序缓冲区的数据高速缓存中，从而留下 200 个数据高速缓存排序缓冲区用于相应开销
- create index 创建一个 50 字节字段，因此每页大约 39 行

因此，将使用所有 5000 个 2K 缓冲区容纳要排序的数据：

(39 X 1,000 X 66) / 2048 = 1,257 个 2K 缓冲区

但是，数据高速缓存还留下 200 个 2K 缓冲区用于相应开销，因此过程高速缓存使用 1057 个 2K 缓冲区。

降低查询处理延迟

Adaptive Server 15.7 中的查询处理层能让多个客户端连接重用或共享动态 SQL 轻量过程 (LWP)。

跨多个连接重用动态 SQL LWP

在 15.7 之前的版本中，Adaptive Server 在动态 SQL 高速缓存中存储动态 SQL 语句 （准备语句）及其相应 LWP。动态 SQL 语句的每个 LWP 都是根据连接元数据确定的。因为连接具有与同一 SQL 语句关联的不同 LWP，所以这些连接不能重用或共享同一 LWP。此外，在释放动态 SQL 高速缓存后，连接创建的所有 LWP 和查询计划都会丢失。

在 15.7 及更高版本中，Adaptive Server 使用语句高速缓存来另行存储转换为 LWP 的动态 SQL 语句。由于语句高速缓存存储在所有连接当中，因此，动态 SQL 语句可以在连接当中重复使用。以下语句不被高速缓存：

- select into 语句。
- 包含所有实际值且不含参数的 insert-values 语句。
- 不引用任何表的查询。
- 包含多个 SQL 语句的各个 prepared 语句。例如：

```
statement.prepare( ‘insert t1 values (1) insert
t2 values (3)’ );
```

- 导致 instead-of 触发器引发的语句。

若要能够使用语句高速缓存存储动态 SQL 语句，请将 enable functionality group 或 streamlined dynamic SQL 配置选项设置为 1。请参见《系统管理指南，卷 1》中的 “设置配置参数”。

使用语句高速缓存提供了诸多好处：

- 当创建了条目的连接存在时，不会从语句高速缓存中清除 LWP 及其关联计划。
- 可以跨各个连接共享 LWP，从而进一步提高性能。
- 重用 LWP 还可以改进 execute 游标的性能。
- 可以从监控表 monCachedStatement 监控动态 SQL 语句。

注释　重用动态 SQL LWP 可能对性能产生负面影响，因为重用计划是使用原来提供的一组参数值生成的。

语句高速缓存

语句高速缓存保存以前为即席 SQL 语句生成的 SQL 文本和计划，这使得 Adaptive Server 能够避免重新编译与以前高速缓存的语句匹配的传入 SQL。启用语句高速缓存后，它能够保留部分过程高速缓存。请参见《系统管理指南，卷 2》，查看语句高速缓存的完整论述，包括其内存使用情况。

数据高速缓存

缺省数据高速缓存和其它高速缓存配置为绝对值。数据高速缓存包含得自最近访问对象的页，通常为：

- sysobjects、 sysindexes 和每个数据库的其它系统表
- 每一数据库的活动日志页
- 常用索引的较高级别和部分较低级别
- 最近访问的数据页

在安装 Adaptive Server 时，有一个供所有 Adaptive Server 进程和对象用于数据、索引和日志页的单独数据高速缓存。缺省大小为 8MB。

以下各页介绍使用此单独数据高速缓存的方法。大部分关于老化、缓冲区清洗和高速缓存策略的概念都适用于用户定义的数据高速缓存和缺省数据高速缓存。

第 90 页的 “配置数据高速缓存以提高性能”介绍如何通过将数据高速缓存划分为命名高速缓存来改进性能，以及如何将特定对象绑定到这些命名高速缓存。

数据高速缓存中的页老化

Adaptive Server 数据高速缓存以 MRU/LRU （最近使用最多的/最近使用最少的）为基础进行管理。高速缓存中的页老化后，进入一个清洗区，在此所有脏页 （在内存中时被修改过的页）都被写入磁盘。有一些例外情况：

- 用宽松 LRU 替换策略配置的高速缓存按上面所述使用清洗区，但不以 MRU/LRU 为基础进行维护。

  通常，清洗区中的页是干净的，即，这些页上的 I/O 已完成。当某个任务或查询从 LRU 端获取一页时，它期望该页是干净的。如果该页不干净，查询必须等待在该页上完成 I/O，这将削弱性能。

- 某特殊策略使索引页和 **OAM 页**比数据页老化得更慢。这些页在某些应用程序被频繁访问，将它们保留在高速缓存中可极大地减少磁盘读取次数。

  有关详细信息，请参见 《系统管理指南，卷 2》中的第 10 章 “检查数据库一致性”。

- Adaptive Server 可能选择使用 LRU 高速缓存替换策略，这种策略不会用整个查询中只使用过一次的页刷新高速缓存中的其它页。

- 检查点进程确保在需要重新启动 Adaptive Server 时，可在合理的时间内完成恢复进程。

  如果检查点进程估计恢复对数据库的更改所花费的时间将超过 recovery interval 配置参数的配置值，它将遍历该高速缓存，将脏页写入磁盘。

- 恢复进程只使用缺省数据高速缓存，以加快该进程。

- 当用户进程之间有空闲时间时，管家清洗任务将脏页写入磁盘。

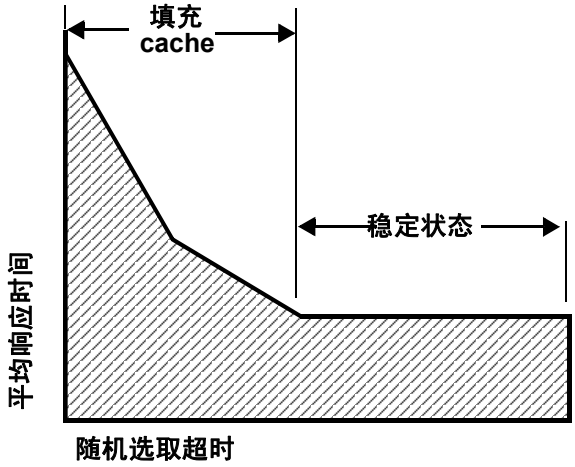
数据高速缓存对检索的影响

图 5-3 显示了数据高速缓存对在一段时间内执行的一系列随机 select 语句的影响。如果高速缓存最初是空的，则第一个 select 肯定能够获得磁盘 I/O。必须确保根据数据库情况，为预期的事务数量确定充分的数据高速缓存大小。

当执行更多查询和填充高速缓存时，高速缓存满足一个或多个页请求的可能性增加，因而该组检索的平均响应时间将缩短。

高速缓存填满后，则肯定能够在高速缓存中找到该点以前的所需页。

图 5-3：随机选取对数据高速缓存的影响



如果高速缓存容量小于所有数据库中被访问的总页数所需的容量，那么给定语句有可能必须执行一些磁盘 I/O。高速缓存不缩短最长可能响应时间 — 某些查询可能仍需要为其所需要的所有页执行物理 I/O。但高速缓存降低了某特定查询经历最大延迟的可能性 — 更多的查询都可能会在高速缓存中至少找到一部分需要的页。

数据修改对高速缓存的影响

存在于更新事务中的高速缓存行为比检索中的更为复杂。

仍有对高速缓存进行填充的初始期段。然后，由于高速缓存页正在被修改，则会有一点，在该点处高速缓存必须开始将这些页写入磁盘，之后才能装载其它页。逐渐地，读写操作量稳定，后续事务有一确定的获取磁盘读取的概率，以及另一导致磁盘写操作的概率。

该稳定状态期段由检查点操作打断，检查点使高速缓存将所有脏页都写入磁盘。

数据高速缓存性能

可通过检查**高速缓存命中率**来观察数据高速缓存的性能，该命中率是由高速缓存处理的页请求的百分比。

百分之百的命中率是非常好的，但意味着您的数据高速缓存与数据等大，或大到至少能包含所有常用表和索引的页。

高速缓存命中率百分比低表明该高速缓存对于当前应用程序负荷来说可能太小。对非常大的表的数据页的随机访问，高速缓存命中率一般比较低。

测试数据高速缓存性能

衡量系统性能时，请考虑数据和过程高速缓存的性能。测试开始时，高速缓存可能处于下列状态之一：

- Empty
- 完全随机化
- 部分随机化
- 确定性的

空的或完全随机化的高速缓存产生重复性的测试结果，因为从一个测试运行到另一个测试时该高速缓存均处于同一状态。

部分随机化或确定性的高速缓存包含刚被执行的事务留下的页。这类页可能是先前测试运行的结果。在此情况下，如果下一测试步骤需要这些页，就不需要磁盘 I/O。

这种情况会使结果偏离纯随机测试，导致性能估计不准。

最佳测试策略是从空高速缓存开始，或确保所有测试步骤都访问数据库的随机部分。为进行精确测试，需要执行与系统上计划的用户混合查询相一致的混合查询。

单个查询的高速缓存命中率

要查看单个查询的高速缓存命中率，请使用 set statistics io on 查看逻辑和物理读取的次数，并使用 set showplan on 查看查询所使用的 I/O 大小。

图 5-4：用于计算高速缓存命中率的公式

$$\text{高速缓存命中率} = \frac{\text{逻辑读取次数} - (\text{物理读取次数} * \text{每 IO 页数})}{\text{逻辑读取次数}}$$

使用 statistics io，物理读取次数以 I/O 大小为单位进行报告。如果查询使用 16K I/O，那么每次 I/O 操作读取 8 页。

如果 statistics io 报告 50 次物理读取，则说明已读取了 400 页。请使用 showplan 查看查询所使用的 I/O 大小。

来自 *sp_sysmon* 的高速缓存命中率信息

sp_sysmon 报告下述项目的高速缓存命中和未命中次数：

- 上的所有高速缓存Adaptive Server
- 缺省数据高速缓存
- 所有用户配置的高速缓存

全服务器范围的报告提供高速缓存搜索的总数，以及高速缓存命中和未命中的百分比。

对于每个高速缓存，该报告都包含高速缓存搜索次数、高速缓存命中和未命中次数，以及在清洗部分找到所需缓冲区的次数。

请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

配置数据高速缓存以提高性能

安装 Adaptive Server 时，它有一个缺省数据高速缓存，其中包含一个 2K 内存池，一个高速缓存分区以及一个螺旋锁。

为提高性能，可添加数据高速缓存，并将数据库或数据库对象绑定到它们：

1 为减少对缺省数据高速缓存螺旋锁的争用，可将高速缓存分成 *n* 个，其中 *n* 为 1、2、4、8、16、32 或 64。如果只有 1 个高速缓存分区时发生对螺旋锁的争用（此处使用“x”指定），那么可期望螺旋锁争用减少 x/*n*，其中 *n* 为分区数。

2 当特定高速缓存分区螺旋锁表为热表 （用户应用程序对其需求较大的表）时，应考虑将缺省高速缓存分割成命名高速缓存。

3 如果仍有争用，则考虑将指定高速缓存分割成指定高速缓存分区。

可根据用户定义的数据高速缓存和缺省数据高速缓存中的逻辑页大小，配置 4K、 8K 和 16K 缓冲池，使 Adaptive Server 能执行大 I/O。另外，大小可完全容纳表或索引的高速缓存可使用宽松的 LRU 高速缓存策略来降低开销。

也可将缺省数据高速缓存或指定高速缓存分割成分区以减少螺旋锁争用。

请尝试通过下列方式配置数据高速缓存以改进性能：

- 配置足够大的命名数据高速缓存以容纳关键表和索引。这样可防止其它服务器活动争用高速缓存空间，并加速使用这些表的查询，因为所需页始终可在高速缓存中找到。

  可配置这些高速缓存以使用能减少高速缓存开销的宽松 LRU 替换策略。

- 为提高并发性，可将热表绑定到一个高速缓存上，而将表上的索引绑定到其它高速缓存。

- 创建足够大的命名数据高速缓存，使其容纳表的热页，而其中大部分查询只引用该表的一部分。

  例如，如果某表包含了一年的数据，但 75% 的查询都引用最近月份的数据 （大约该表的 8%），则配置一个大小为该表 10% 的高速缓存，以将最常用的页保存在该高速缓存中，并为较不常用页留出一些空间。

- 将决策支持系统 (DSS) 使用的表或数据库分配到已配置了大 I/O 的特定高速缓存中。

  这样可防止 DSS 应用程序与 OLTP 应用程序争用高速缓存空间。DSS 应用程序通常访问大量连续页，而 OLTP 应用程序通常访问相对较少的随机页。

- 将 tempdb 绑定到其自己的高速缓存上，防止它与其它用户进程发生争用。

  适当确定 tempdb 高速缓存的大小可为许多应用程序将大部分 tempdb 活动保留在内存中。如果此高速缓存足够大，则 tempdb 活动可避免执行 I/O。

- 将文本页绑定到命名高速缓存，以提高文本访问的性能。
- 将数据库日志绑定到高速缓存，再次减少对高速缓存空间和高速缓存访问的争用。
- 当用户进程对高速缓存进行更改后，螺旋锁会拒绝所有其它进程对此高速缓存的访问。

  虽然螺旋锁的持有时间极短，也会减慢具有高事务处理速率的多处理器系统的性能。配置多个高速缓存时，每个高速缓存都由单个螺旋锁控制，提高具有多个 CPU 的系统的并发性。

  在单个高速缓存中，添加高速缓存分区可创建多个螺旋锁，进一步减少争用。单引擎服务器不存在螺旋锁争用问题。

指定数据高速缓存的这些可能用途中的大部分，对具有高事务处理速率或频繁 DSS 查询和多用户的多处理器系统的影响最大。其中一些可提高单 CPU 系统的性能，在它们能提高内存使用率并减少 I/O 时。

配置指定数据高速缓存的命令

用于配置高速缓存和缓冲池的命令如所示表 5-1。

表 5-1：用于配置高速缓存的命令

命令	快捷键
sp_cacheconfig	创建或删除指定高速缓存，并设置大小、高速缓存类型、高速缓存策略或本地高速缓存分区数量。报告高速缓存和缓冲池大小。
sp_poolconfig	创建和删除 I/O 缓冲池，并更改它们的大小、清洗大小和异步预取限制。
sp_bindcache	将数据库或数据库对象绑定到高速缓存上。
sp_unbindcache	解除指定数据库或数据库对象与高速缓存的绑定。

命令	快捷键
sp_unbindcache_all	解除绑定到指定高速缓存的所有数据库和对象。
sp_helpcache	报告数据高速缓存的摘要信息并列出绑定到高速缓存的数据库和数据库对象。还报告高速缓存所需的开销数量。
sp_sysmon	报告用于调优高速缓存配置的统计信息，包括高速缓存螺旋锁争用、高速缓存利用率和磁盘 I/O 模式。

有关配置命名高速缓存以及将对象绑定到高速缓存的完整说明，请参见《系统管理指南，卷 2》中的第 4 章 “配置数据高速缓存”。只有系统管理员才能配置命名高速缓存，并将数据库对象绑定到它们。

调优指定高速缓存

创建命名数据高速缓存和内存池，以及将数据库和数据库对象绑定到这些高速缓存能显著降低或提高 Adaptive Server 性能。例如：

- 高速缓存使用不当会损害性能。

  如果为服务器上只处理很小比例查询活动的数据库分配 25% 的数据高速缓存，那么其它高速缓存上的 I/O 会增加。

- 未使用的池会降低性能。

  如果添加 16K 缓冲池，但没有任何查询使用它，那么就已经占用了 2K 缓冲池的空间。 2K 缓冲池的高速缓存命中率降低， I/O 增加。

- 过度使用的池会降低性能。

  如果配置一个 16K 的池，而实际上所有查询都使用该缓冲池，则 I/O 率会增加。页在 16K 缓冲池中快速循环，而 2K 高速缓存的利用率却不足。 16K 缓冲池中的高速缓存命中率将非常低。

- 在平衡高速缓存中缓冲池的利用率后，性能可极大提高。

  16K 和 2K 查询的高速缓存命中率都得到提高。执行 16K I/O 的查询经常使用的大量页不会刷新磁盘中的 2K 页。使用 16K 的查询执行的 I/O 数量大约相当于 2K I/O 所需 I/O 的八分之一。

调优指定高速缓存时，应始终测量当前性能，进行配置更改，并在相同工作量下测量这些更改的效果。

高速缓存配置目标

配置高速缓存的目标包括：

- 在多引擎服务器上，减少对螺旋锁的争用。
- 提高高速缓存命中率和降低磁盘 I/O。作为回报，提高查询的高速缓存命中率可减少锁争用，因为不需要执行物理 I/O 的查询通常只能在较短时间内持有锁。
- 由于高效利用大 I/O，物理读取次数减少。
- 减少物理写入次数，因为其它进程不会从高速缓存中清除最近修改的页。
- 适当使用宽松 LRU 策略时，高速缓存开销减少且 SMP 系统上的 CPU 总线延迟时间缩短。
- 使用高速缓存分区时， SMP 系统上对高速缓存螺旋锁的争用减少。

除了以每查询为基础帮助调优的命令 （如 showplan 和 statistics io ）外，还需要使用性能监控工具 （如 sp_sysmon）来了解多个查询和多个应用程序是如何在同时运行时共享高速缓存空间的。

收集数据并制定计划，然后执行

制定高速缓存使用计划的第一个步骤是为数据高速缓存提供尽可能多的内存：

- 确定可分配给 Adaptive Server 的最大内存量。将 max memory 设置为该值。
- 设置使用 Adaptive Server 内存的所有参数后， max memory 和 total logical memory 的运行值之间的差值即为可用于其它配置及数据和过程高速缓存的内存量。如果已充分配置其它所有配置参数，那么可将此附加内存分配给数据高速缓存。对数据高速缓存进行的大多数更改是动态的并且不需要重新启动。
- 如果将所有附加内存都分配给数据高速缓存，那么就没有用于重新配置其它配置参数的可用内存了。但是，如果有额外内存可用，则可动态增加 max memory 和其它动态配置参数（例如 procedure cache size、 user connections 等）。
- 使用性能监控工具确定基本性能及调优目标。

可根据前面的步骤确定可分配给数据高速缓存的内存大小。包括已配置高速缓存的大小，如缺省数据高速缓存和所有命名高速缓存。

通过查看现有对象和应用程序可确定数据高速缓存的大小。添加新高速缓存或增加消耗内存的配置参数不会减小缺省数据高速缓存的大小。在确定数据高速缓存可用的内存和单个高速缓存的大小后，可添加新高速缓存和增加或减少现有数据高速缓存的大小。

- 通过分析 I/O 模式估计高速缓存需求，并通过分析查询计划和 I/O 统计信息估计缓冲池需求。
- 首先配置能获得最佳性能的、最简单的方案：
  - 为 tempdb 高速缓存选择大小。
  - 为所有日志高速缓存选择大小，并调优日志 I/O 大小。
  - 为要完整保留在高速缓存中的特定表或索引选择大小。
  - 为索引或数据高速缓存添加大 I/O 缓冲池 （如果适当）。
- 确定这些大小后，检查其余 I/O 模式、高速缓存争用情况和查询性能。按 I/O 使用的比例为对象和数据库配置高速缓存。

配置高速缓存时，要牢记性能目标：

- 如果主要目标是减少螺旋锁争用，那么增加频繁使用的高速缓存的分区数可能是唯一的方法。

  将少数几个高 I/O 对象移动到单独的高速缓存中也可以减少螺旋锁争用并提高性能。
- 如果主要目标是通过提高特定查询或应用程序的高速缓存命中率来缩短响应时间，那么在为这些查询使用的表和索引创建高速缓存之前，应全面了解访问方法和 I/O 要求。

估计高速缓存需求

通常，可根据查询将访问高速缓存中页的次数按比例配置高速缓存，并根据选择了某种缓冲池大小的 I/O 的查询所使用的页数按比例配置高速缓存中的缓冲池。

如果数据库及其日志位于单独的逻辑设备上，则可使用 sp_sysmon 或操作系统命令估计高速缓存比例，以便按设备检查物理 I/O。

请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

大 I/O 和性能

通过将高速缓存分割成缓冲池，可配置缺省高速缓存和所有为大 I/O 创建的指定高速缓存。缺省 I/O 大小为 2K，即一个 Adaptive Server 数据页。

注释　对大 I/O 的引用位于一台逻辑页大小为 2K 的服务器上。如果服务器的逻辑页大小为 8K，则用于 I/O 的基本单位为 8K。如果服务器的逻辑页大小为 16K，则用于 I/O 的基本单位为 16K。

对于频繁存储和访问页的查询， Adaptive Server 在单个 I/O 中最多读取八个数据页。由于大部分 I/O 时间都花在磁盘上的物理定位和搜寻上，因此大 I/O 可极大地缩短磁盘访问时间。在大多数情况下，可在缺省数据高速缓存中配置一个 16K 缓冲池。

某些类型的 Adaptive Server 查询有可能从大 I/O 中获益。识别这些类型的查询有助于确定数据高速缓存和内存池的正确大小。

在下列示例中，要么数据库或特定的表、索引或大对象 LOB 页更改（用于 text、image 和 Java 行外列）都必须绑定到具有大内存池的命名数据高速缓存，要么缺省数据高速缓存必须有大 I/O 缓冲池。可从大 I/O 中获益的查询类型包括：

- 扫描整个表的查询。例如：

```
select title_id, price from titles
select count(*) from authors
    where state = "CA"   /* no index on state */
```

- 具有聚簇索引的表上的范围查询。例如：

```
where indexed_colname >= value
```

- 扫描索引的叶级的查询，包括匹配和非匹配扫描。如果在 type、price 上有非聚簇索引，因为查询中使用的所有列都包含在该索引中，所以此查询可使用索引叶级上的大 I/O：

```
select type, sum(price)
    from titles
    group by type
```

- 连接整个表或表的大部分的查询。不同的 I/O 大小可用在一个连接的不同表中。
- 选择 text 或 image 或 Java 行外列的查询。例如：

```
select au_id, copy from blurbs
```

- 生成笛卡尔乘积的查询。例如：

```
select title, au_lname
     from titles, authors
```

  此查询需要扫描一个表的全部，并为第一个表的每一行完全扫描其它表。这些查询的高速缓存策略遵循与连接相同的原则。

- 像 select into 这样分配大量页的查询。

  注释 Adaptive Server 版本 12.5.0.3 或更高版本在 select into 中启用大页分配。它根据扩充而不是单个页来分配页，因此为目标表发出更少的日志请求。

  如果为 Adaptive Server 配置大缓冲池，那么它会在将目标表页写入磁盘时使用大 I/O 缓冲池。

- create index 命令。
- 堆上的批量复制操作 — 拷入和拷出。
- update statistics、 dbcc checktable 和 dbcc checkdb 命令。

优化程序和高速缓存选择

如果表或索引的高速缓存有 16K 缓冲池，那么优化程序将根据需要读取的页数和表或索引的集群比，确定用于数据和叶级索引页的 I/O 大小。

优化程序的信息仅限于它所分析的单个查询，以及关于表和高速缓存的统计信息。它没有关于有多少其它查询正在同时使用同一数据高速缓存的信息。也没有关于分割表存储的方式是否将使大 I/O 或异步预取效率降低的统计信息。

在某些情况下，这种不同因素的组合会导致产生过多 I/O。例如，如果具有大型结果集的同步查询使用很小的内存池，将出现较高的 I/O 和较低的性能。

为高速缓存选择合适的 I/O 大小组合

在任意数据高速缓存中，可最多配置四个缓冲池，但在大多数情况下，单个对象的高速缓存在只使用一个 2K 缓冲池和一个 16K 缓冲池时能获得最佳性能。日志未绑定到单独高速缓存的数据库的高速缓存也应有一个配置为与日志 I/O（为该数据库配置的）大小相匹配的缓冲池；通常，最佳的日志 I/O 大小为 4K。

减少与高速缓存分区的螺旋锁争用

随着 SMP 系统上运行的引擎和任务数量增加时，对数据高速缓存上螺旋锁的争用也增加。任何时候，某个任务需要访问高速缓存查找其中的页或重新链接 LRU/MRU 链上的页时，它都持有高速缓存螺旋锁，以防止其它任务同时修改该高速缓存。

有多个引擎和用户时，任务必须等待才能访问高速缓存。添加高速缓存分区可将高速缓存分成各个分区，每个分区都由各自的螺旋锁提供保护。当需要将某页读入高速缓存或对其进行定位时，将对数据库 ID 和页 ID 应用散列函数，以确定包含该页的分区。

高速缓存分区数始终为 2 的乘方。每次增加分区数时，都会将螺旋锁争用减少约 1/2。如果螺旋锁争用大于 10% 到 15%，应考虑增加该高速缓存的分区数量。本示例在缺省数据高速缓存中创建了 4 个分区：

```
sp_cacheconfig "default data cache",
"cache_partition=4"
```

必须重新启动服务器才能使高速缓存分区中的更改生效。

请参见 《系统管理指南，卷 2》中的第 4 章 “配置数据高速缓存”。

有关使用 sp_sysmon 监控高速缓存螺旋锁争用的信息，请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

高速缓存中的每一个缓冲池都被分区成页的单独 LRU/MRU 链，有其自己的清洗标记。

高速缓存替换策略和政策

Adaptive Server 优化程序使用两种高速缓存替换策略在刷新不常用页时将常用页保留在高速缓存中。为降低高速缓存开销，可能需要考虑为某些高速缓存设置高速缓存替换策略。

策略

替换策略决定从磁盘中读取页时，将其放在高速缓存的什么位置。优化程序决定每个查询所使用的高速缓存替换策略：

- 读取和放弃 （或 MRU 替换）策略可在缓冲池中的清洗标记处链接新读取的缓冲区。
- LRU 替换策略可在缓冲池的最近使用 (MRU) 端链接新读取的缓冲区。

高速缓存替换策略可影响查询混合的高速缓存命中率：

- 用读取和放弃策略读入高速缓存的页保留在高速缓存中的时间比在高速缓存 MRU 端读入的查询短很多。如果再次需要此类页 （例如，如果很快又运行相同的查询），那么这些页可能需要再次从磁盘中读取。
- 用读取和放弃策略读入高速缓存的页不替换已驻留在高速缓存中清洗标记前的页。这意味着已驻留在高速缓存中清洗标记前的页不会被查询只使用过一次的页从高速缓存中清出。

请参见 《性能和调优系列：查询处理和抽象计划》中的第 7 章 “控制优化”。

政策

系统管理员可指定是否要将高速缓存作为 MRU/LRU 链接页列表进行维护 （严格）或是否可使用宽松 LRU 替换策略。

- **严格替换策略** — 替换缓冲池中最近最少使用的页，链接缓冲池中页链开始处 （MRU 端）新读取的页。
- **宽松替换策略** — 尝试避免替换最近使用的页，但不产生使缓冲区保持 LRU/MRU 顺序的开销。

缺省高速缓存替换政策是严格替换。仅在同时满足以下两个条件时使用宽松替换策略：

- 高速缓存中极少或根本没有缓冲区替换。
- 数据从不更新或极少更新。

宽松 LRU 政策节省了使高速缓存保持 MRU/LRU 顺序的开销。在高速缓存页的副本可以驻留在 CPU 本身的硬件高速缓存中的 SMP 系统上，宽松 LRU 政策可减少连接 CPU 的总线的带宽。

如果已创建了一个高速缓存来容纳所有或大部分特定对象，且高速缓存命中率大于 95%，那么对此高速缓存使用宽松高速缓存替换策略能略微提高性能。

请参见 《系统管理指南，卷 2》中的第 4 章 “配置数据高速缓存”。

为数据库日志配置宽松 LRU 替换

日志页被日志记录所填充，并立即写入磁盘。当应用程序包括触发器、延迟更新或事务回退时，可能要读取一些日志页，但它们通常都是近期用过的页，仍在高速缓存中。

由于访问高速缓存中的这些页会将它们移到严格替换政策高速缓存的 MRU 端，因此使用宽松 LRU 替换时，日志高速缓存可能性能更好。

查寻表和索引的宽松 LRU 替换

大小确定为可容纳索引和常用查寻表的用户定义的高速缓存是宽松 LRU 替换的适用对象。如果某高速缓存是适用对象，但高速缓存命中率稍低于 95% 这一性能指导值，则要确定略微增加高速缓存的大小是否能提供足够空间来完全容纳表或索引。

指定数据高速缓存建议

这些高速缓存建议既可提高单处理器服务器性能，又可提高多处理器服务器性能：

- 因为 Adaptive Server 根据逻辑页的大小写入日志页，所以较大的日志页可能会降低对日志页的提交共享写入率。

  当 Adaptive Server 等到可一次执行一批提交（而不是执行许多单个提交）时，会发生提交共享。每个进程的用户日志高速缓存的大小根据逻辑页大小和 user log cache size 配置参数确定。用户日志高速缓存的缺省大小是一个逻辑页。

  对于生成很多日志记录的事务，刷新用户日志高速缓存所需的时间比刷新较大逻辑页所需的时间要略微长一些。当然，因为日志高速缓存大小也较大， Adaptive Server 不需要像许多日志高速缓存一样为长期事务刷新到日志页。

  请参见《系统管理指南，卷 2》中的第 4 章“配置数据高速缓存”。

- 为 tempdb 创建一个命名高速缓存，并将该高速缓存配置为 16K I/O，以供 select into 查询和排序使用。

- 为高使用率数据库的日志创建命名高速缓存。配置此高速缓存中的缓冲池以与用 sp_logiosize 设置的日志 I/O 大小相匹配。

  请参见第 102 页的 “为事务日志选择 I/O 大小”。

- 如果表或其索引较小且一直在使用，则仅为该对象或该少数对象创建高速缓存。
- 对于高速缓存命中率高于 95% 的高速缓存，如果使用多个引擎，则配置宽松 LRU 高速缓存替换政策。
- 根据高速缓存使用对象和查询按比例保留高速缓存和缓冲池的大小：
  - 如果服务器上 75% 的工作都在一个数据库上执行，那么在特别为该数据库创建的高速缓存中、在为其最繁忙的表和索引创建的高速缓存中、或在缺省数据高速缓存中，就应为该数据库分配大约 75% 的数据高速缓存。
  - 如果数据库中大约 50% 的工作可使用大 I/O，则将大约 50% 的高速缓存按 16K 内存池配置。
- 应将高速缓存作为共享资源来查看，而不是尝试细致地管理每个表和索引的高速缓存需求。

  启动高速缓存分析，在数据库级别进行测试，专注于有高 I/O 需求或高应用程序优先权的个别表和对象，以及那些有特殊用法的项目，如 tempdb 和事务日志。
- 在 SMP 服务器上，使用多个高速缓存以避免对高速缓存螺旋锁的争用：
  - 为繁忙数据库的事务日志使用单独的高速缓存，为经常访问的一些表和索引使用单独的高速缓存。
  - 如果某高速缓存上螺旋锁争用大于 10%，则将其分割成多个高速缓存或使用高速缓存分区。

    在高使用率期间定期执行 sp_sysmon，以检查高速缓存争用情况。请参见 《性能和调优系列：使用 with sp_sysmon 监控 Adaptive Server》。
  - 为高速缓存命中率在 95% 以上的高速缓存设置宽松 LRU 高速缓存政策，如为容纳整个表或索引而配置的高速缓存。

确定用于特殊对象、 *tempdb* 和事务日志的高速缓存大小

为 tempdb、事务日志、要完整保存在高速缓存中的几个表或索引创建高速缓存，可减少高速缓存螺旋锁争用并提高高速缓存命中率。

可使用 sp_spaceused 来确定要整个保留在高速缓存中的表或索引的大小。如果知道这些表的大小的增加速度，则为它们的增长留出一些额外的高速缓存空间。要查看表的所有索引的大小，使用：

```
sp_spaceused table_name, 1
```

检查 *tempdb* 的高速缓存需求

查看 tempdb 的使用情况：

- 估计查询所生成的临时表和工作表的大小。

  查看 select into 查询生成的页数。

  这些查询可使用 16K I/O，因此，使用此信息可帮助您确定 tempdb 高速缓存的 16K 缓冲池的大小。

- 估计临时表和工作表的持续时间 （按挂钟时间）。
- 估计创建临时表和工作表的查询的执行频率。
- 尝试估计同时用户的数量，特别是对那些在 tempdb 中生成很大结果集的查询。

根据此信息，可对 tempdb 中同时发生的 I/O 活动的数量做出粗略估计。根据您的其它高速缓存需要，可确定 tempdb 的大小，以使所有 tempdb 活动都实际发生在高速缓存中，并且几乎没有要实际写入磁盘的临时表。

在大多数情况下， tempdb 的前 2MB 空间存储在 master 设备中，其余空间分配给了另一台逻辑设备。可使用 sp_sysmon 检查这些设备以帮助确定物理 I/O 率。

检查事务日志的高速缓存需求

在具有高事务处理速率的 SMP 系统上，将事务日志绑定到其自己的高速缓存可减少缺省数据高速缓存上的高速缓存螺旋锁争用。在很多情况下，日志高速缓存非常小。

在提交事务时，事务日志的当前页会写入磁盘，所以可尝试确定日志大小以减少需要重新读取日志页的进程必须访问磁盘的次数，因为已从高速缓存中清除这些页。

以下 Adaptive Server 进程需要读取日志页：

- 使用 inserted 和 deleted 表的触发器 （这些表是在触发器查询它们时根据事务日志创建的）
- 延迟更新、删除和插入，因为这些查询都需要重新读取日志以便将更改应用到表或索引中
- 回退事务，因为必须访问日志页以回退更改

为事务日志确定高速缓存大小时：

- 检查需要重新读取日志页的进程的持续时间。

  估计最长的触发器和延迟更新持续的时间有多长。

  如果某些运行时间长的事务被回退，则检查它们已运行的时间长度。

- 可使用 sp_spaceused 定期检查事务日志大小，以估计日志的增长速度。

用此日志增长估计和时间估计来确定日志高速缓存的大小。例如，如果最长的延迟更新需要 5 分钟，而此数据库的事务日志每分钟增长 125 页，则在执行此事务时，为该日志分配了 625 页。

如果少数几个事务或查询运行的时间特别长，那么可能要根据平均时间长度 （而非最大时间长度）来确定日志大小。

为事务日志选择 I/O 大小

当用户执行需要日志记录的操作时，日志记录首先存储在用户日志高速缓存中，直到某些事件将用户的日志记录刷新到高速缓存中的当前事务日志页。在以下情况下将刷新日志记录：

- 事务结束时
- 用户日志高速缓存已填满
- 事务更改了另一数据库中的表
- 另一进程需要写入用户日志高速缓存引用的页
- 发生了某些系统事件

为了减少磁盘写入， Adaptive Server 将部分填充的事务日志页保留一段较短的时间，以便可将几个事务的记录同时写入磁盘。这一过程称为分组提交。

在事务处理速率高或具有创建大日志记录的事务的环境中， 2K 事务日志页将被迅速填满。大部分日志写入操作是日志页填满引起的，而不是分组提交。

为事务日志创建 4K 缓冲池可极大地减少这些环境中日志写操作的数量。

sp_sysmon 报告事务日志写操作与事务日志分配的比率。如果满足以下所有条件，则应尝试使用 4K 日志 I/O：

- 数据库使用 2K 日志 I/O。
- 每秒钟的日志写操作数量较高。
- 每日志页的平均写入操作数略微大于 1。

这里是一些显示较大的日志 I/O 大小可能有助于提高性能的示例输出：

```
                         per sec   per xact    count  % of total
Transaction Log Writes      22.5      458.0     1374     n/a
Transaction Log Alloc       20.8      423.0     1269     n/a
Avg # Writes per Log Page    n/a        n/a  1.08274     n/a
```

请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

配置为大日志 I/O

Adaptive Server 启动时，服务器错误日志中将报告每个数据库的日志 I/O 大小。也可使用 sp_logiosize。

要查看当前数据库的大小，请不带参数执行 sp_logiosize。要查看服务器上所有数据库的大小和日志使用的高速缓存的大小，请使用：

```
sp_logiosize "all"
```

要将数据库的日志 I/O 大小设置为 4K,（缺省值），必须正在使用该数据库。此命令将大小设置为 4K：

```
sp_logiosize "default"
```

如果在日志所用的高速缓存中无 4K 缓冲池，就用 2K I/O 代替。

如果数据库绑定到高速缓存，则所有未明确绑定到其它高速缓存的对象都使用该数据库的高速缓存。其中包括 syslogs 表。

要将 syslogs 绑定到另一高速缓存，必须首先用 sp_dboption 将数据库置于单用户模式，然后使用该数据库并执行 sp_bindcache：

```
sp_bindcache pubs_log, pubtune, syslogs
```

日志高速缓存的其它调优技巧

要在配置日志高速缓存后进一步调优，可检查以下项的 sp_sysmon 输出：

- 日志所用的高速缓存
- 日志存储于其上的磁盘
- 每日志页的平均写操作次数

查看日志高速缓存部分时，检查 “Cache Hits”和 “Cache Misses”以确定是否能在高速缓存中找到延迟操作、触发器和回退所需的大部分页。

在 “Disk Activity Detail”部分中，观察已执行的 “Reads”数，以查看需要重新读取日志的任务有多少次不得不访问了磁盘。

以查询计划和 I/O 为基础配置数据缓冲池大小

基于使用相应 I/O 大小的查询所执行的 I/O 比例，将高速缓存分成多个缓冲池。如果大多数查询都可从 16K I/O 中获益，而您配置了一个非常小的 16K 高速缓存，则性能会下降。

2K 缓冲池中的大部分空间保留未用，而 16K 缓冲池周转率非常高。高速缓存命中率显著降低。

这个问题对于嵌套循环连接查询最为严重，因为这些查询必须反复从磁盘重新读取内部表。

选择合适的缓冲池大小要求：

- 了解应用程序混合知识和查询可用的 I/O 大小
- 使用监控工具检查高速缓存利用率、高速缓存命中率和磁盘 I/O，仔细研究并调优。

检查查询的 I/O 大小

可检查查询计划和 I/O 统计信息以确定哪些查询可能执行大 I/O，以及这些查询所执行的 I/O 数量。可以这些信息为基础，估计查询应使用 16K 内存池执行的 16K I/O 的数量。I/O 按逻辑页大小完成，如果大 I/O 使用 2K 页，则按 2K 大小进行检索，如果使用 8K 页，则按 8K 大小进行检索，如下所示：

逻辑页大小	内存池
2K	16K
4K	32K
8K	64K
16K	128K

另一个示例是，使用 2K 缓冲池扫描表并执行 800 次物理 I/O 的查询应执行大约 100 次 8K I/O。

有关查询类型列表，请参见第 95 页的 “大 I/O 和性能”。

为测试您的估计，需要配置缓冲池并运行单个查询和目标混合查询，以确定最佳缓冲池大小。为第一个使用 16K I/O 的测试选择的初始大小是否合适，取决于对应用程序混合中查询类型的了解程度。

如果是在一个活动生产服务器上第一次配置 16K 缓冲池，则此估计尤为重要。尽可能地准确估计对高速缓存的同时使用情况。

以下指南提供了一些参考点：

- 如果大多数 I/O 发生在使用索引访问少量行的点查询中，则使 16K 缓冲池相对小一些，大约为高速缓存大小的 10% 到 20%。
- 如果估计大部分 I/O 都将使用 16K 缓冲池，则将高速缓存的 50% 到 75% 配置给 16K I/O。

  使用 16K I/O 的查询包括扫描表、使用聚簇索引进行范围搜索和 order by 的所有查询，以及对覆盖非聚簇索引执行匹配或非匹配扫描的查询。
- 如果您不能确定查询要使用的 I/O 大小，应按 16K 缓冲池将其配置为高速缓存空间的大约 20%，并在运行查询时使用 showplan 和 statistics i/o。

  检查 showplan 输出，查找 “Using 16K I/O” 消息。检查 statistics i/o 输出以查看执行了多少 I/O。
- 如果认为典型的应用程序混合同时使用 16K I/O 和 2K I/O，那么为 16K I/O 配置高速缓存空间的 30% 到 40%。

  根据实际混合和查询所选择的 I/O 大小，您的最佳配置可能与此不同。

  如果同时通过 2K I/O 和 16K I/O 访问许多表，且来自扩充的所有页都在 2K 高速缓存中，那么 Adaptive Server 不能使用 16K I/O。它对查询所需的扩充内的其它页执行 2K I/O。这将添加到 2K 高速缓存的 I/O。

  为 16K I/O 进行配置后，检查高速缓存使用情况并监控受影响设备的 I/O （使用 sp_sysmon 或 Adaptive Server Monitor）。同样，使用 showplan 和 statistics io 来查看查询。

  - 寻找其中的内部表将使用 16K I/O 的嵌套循环连接查询，并用读取和放弃 (MRU) 策略重复扫描表。

    当高速缓存无法完全容纳外部表或内部表时，会发生这种情况。通过增加 16K 缓冲池的大小以使高速缓存完全容纳内部表，可极大地减少 I/O。也可以考虑将这两个表绑定到单独的高速缓存。
  - 与页上的表大小比较，寻找多余的 16K I/O。

    例如，如果有一个 8000 页的表，而 16K I/O 的表扫描执行了大大超过 1000 次 I/O 来读取此表，那么通过在此表上重新创建聚簇索引可使性能提高。
  - 寻找大 I/O 被拒绝的次数。很多情况下，这是因为页已经位于 2K 缓冲池中，所以 2K 缓冲池将用于该查询的其余 I/O。

请参见 《性能和调优系列：查询处理和抽象计划》中的第 7 章“控制优化”。

配置缓冲区清洗大小

可为每个高速缓存中的每个缓冲池配置清洗区域。如果将清洗大小设置得太高， Adaptive Server 可能要执行不必要的写入。如果将清洗区域设置得太小，Adaptive Server 可能无法在缓冲区链末端找到干净的缓冲区，因而可能必须等到 I/O 完成后才能继续操作。通常，清洗大小缺省值都是正确的，仅在具有高数据修改率的大缓冲池中才需要调整。

Adaptive Server 以逻辑页为单位分配缓冲池。例如，在使用 2K 逻辑页的服务器上，为缺省数据高速缓存分配 8MB。缺省情况下，这会构成大约 4096 个缓冲区。

如果将相同的 8MB 分配给使用 16K 逻辑页大小的服务器上的缺省数据高速缓存，那么缺省数据高速缓存大约为 512 个缓冲区。在繁忙系统上，此小数量的缓冲区可导致缓冲区始终处于清洗区，减慢了要求干净缓冲区的任务的执行速度。

通常，若要在使用较大页大小时获得与 2K 逻辑页大小相同的缓冲区管理特性，请根据较大的页大小来确定高速缓存的大小。即，如果将逻辑页大小增大为原来的四倍，那么高速缓存和缓冲池大小也应为原来大小的四倍。

执行大 I/O、基于扩充的读写操作的查询使用较大逻辑页大小更有益。当然，当目录不断成为锁定页时，在系统目录的页级上会有更大的争用和锁定。

在使用宽列时，对 DOL 锁定表执行行和列复制操作会进一步降低服务器速度。为支持宽行和宽列而进行的内存分配将或多或少地减慢服务器速度。

请参见 Performance and Tuning Series: Locking and Concurrency Control（《性能和调优系列：锁定和并发控制》）。

缓冲池配置和绑定对象的开销

配置内存池以及将对象绑定到高速缓存会对生产系统上的用户造成影响，因此最好在非高峰时间执行这些活动。

缓冲池配置开销

创建、删除或更改缓冲池时，所有使用绑定到高速缓存的对象的存储过程和触发器的计划都在下次运行时重新编译。如果数据库绑定到高速缓存，这将会影响数据库中的所有对象。

在缓冲池之间移动缓冲区需要少量开销。

高速缓存绑定开销

当绑定或解除绑定对象时，在绑定进程中，当前存在于高速缓存中的所有对象页都被刷新到磁盘 （如果脏），或从高速缓存中删除 （如果干净）。

用户查询下次需要这些页时，必须再次从磁盘中读取，这就降低了查询性能。

清理高速缓存时， Adaptive Server 获取表或索引的排它锁，因此绑定会减慢其他用户访问对象的速度。绑定进程可能要一直等到事务完成才能获取锁。

注释　将对象绑定到高速缓存或将其解除绑定将从内存中删除这些对象。当您在开发和测试期间优化查询时，这可能很有用。

如果需要检查特定表的物理 I/O，且早期调优的结果已将页引入高速缓存，那么可解除绑定并重新绑定对象。下次访问该表时，查询所用的所有页都必须读入高速缓存。

所有使用绑定对象的存储过程和触发器的计划都在下次运行时重新编译。如果数据库绑定到高速缓存，这将影响数据库中的所有对象。

维护大 I/O 的数据高速缓存性能

刚刚创建堆表、聚簇索引或非聚簇索引后，使用大 I/O 会使它们的性能最佳。一段时间之后，删除、页拆分、页解除分配和重新分配的影响会增加 I/O 消耗。optdiag 将报告有关表和索引的称为“Large I/O efficiency”的统计信息。

当此值为 1 或接近 1 时，大 I/O 非常有效。当此值减小时，需要更多的 I/O 才能访问查询所需的数据页，并且大 I/O 可能将查询不需要的页引入高速缓存中。

当大 I/O 效率下降或缓冲池中的活动由于增加的 16K I/O 而增加时，需要考虑重新创建索引。

当大 I/O 效率下降时，可执行以下操作：

- 在使用仅数据锁定的表上运行 reorg rebuild。也可在 DOL 锁定表的索引上使用 reorg rebuild。
- 对于所有页锁定表，删除并重新创建索引。

请参见 《性能和调优系列：物理数据库调优》中的第 6 章 “数据库维护”。

诊断过多的 I/O 计数

如下原因可以解释为什么执行大 I/O 的查询需要的读取次数比预计的多：

- 查询所用的高速缓存有一 2K 高速缓存，其它进程已将页从表中引入该 2K 高速缓存。

  如果 Adaptive Server 发现要使用 16K I/O 读取的页之一已在 2K 高速缓存中，它将对扩充内查询所需的其它页执行 2K I/O。

- 在每一分配单元的第一个扩充存储了分配页，所以如果查询需要访问该扩充内的所有页，它就必须对与分配页一起共享该扩充的 7 页执行 2K I/O。

  可以使用 16K I/O 读取其它 31 个扩充。整个分配单元的最小读取数始终为 38，而不是 32。

- 在 DOL 锁定表的非聚簇索引和聚簇索引中，扩充既可以存储叶级页又可以存储索引的较高级别中的页。 2K I/O 在索引的较高级别上执行，且当查询需要很少的页时对叶级页执行。

  当覆盖叶级扫描执行 16K I/O 时，一些扩充中的一些页可能在 2K 高速缓存中。扩充中的其余页将用 2K I/O 读取。

使用 *sp_sysmon* 检查大 I/O 性能

每个数据高速缓存的 sp_sysmon 输出都包括可帮助确定大 I/O 效率的信息。请参见《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》，以及《性能和调优系列：查询处理和抽象计划》中的第 7 章“控制优化”。

- “大 I/O 使用情况”报告已执行或已拒绝的大 I/O 的数量，并提供汇总统计信息。
- “大 I/O 详细信息”报告通过大 I/O 读入高速缓存的总页数，以及当这些页位于高速缓存中时的实际访问页数。

恢复速度

用于在 Adaptive Server 中修改数据时，只有事务日志会立即写入磁盘，以确保可以恢复给定数据或事务。更改的或“脏”数据和索引页保留在数据高速缓存中，直到以下事件之一使它们被写入磁盘：

- 检查点进程被唤醒，确定特定数据库的已更改数据和索引页需要写入磁盘，并将数据库所用每一高速缓存中的所有脏页写出。

  recovery interval 的设置和服务器的数据修改率共同决定检查点进程将已更改页写入磁盘的频率。
- 页移入高速缓存的缓冲区清洗区时，脏页被自动写入磁盘。
- Adaptive Server 在用户事务处理过程中具有空闲 CPU 周期和磁盘 I/O 能力，管家任务利用此时间将脏缓冲区写入磁盘。
- 恢复只发生在缺省数据高速缓存上。
- 用户发出一个 checkpoint 命令。

  可使用 checkpoint 确定一个或多个数据库，也可使用 all 子句。

  ```
  checkpoint [all | [dbname[, dbname[, dbname.....]]]
  ```

检查点、管家、以及在清洗标记处开始的写入结合起来具有以下优点：

- 很多事务可以更改高速缓存中的页，或读取高速缓存中的该页，但只执行一次物理写入。
- Adaptive Server 在 I/O 不会引起与用户进程进行争用时执行大量物理写入。

调优恢复间隔

Adaptive Server 的缺省恢复间隔为每个数据库五分钟。更改恢复间隔会影响性能，因为它会影响 Adaptive Server 将页写入磁盘的次数。

表 5-2 显示了更改系统上恢复间隔当前设置所产生的影响。

表 5-2：恢复间隔对性能和恢复时间的影响

设置	对性能的影响	对恢复的影响
较低	可能导致更多的读写操作，并可能降低吞吐量。 Adaptive Server 将更频繁地将脏页写入磁盘。所有检查点 I/O 测试信号将更弱。	如果没有 Adaptive Server 必须回退的长时间运行的未完结事务，那么将恢复间隔设置得较短会加速恢复。 如果存在长时间运行的未完结事务，那么更频繁的执行检查点进程会降低恢复进程的速度，因为磁盘包含 Adaptive Server 必须回退的多个修改内容。
较高	最小化写入操作并提高系统吞吐量。检查点 I/O 测试信号会更强。	自动恢复可能要花更多时间来启动。 Adaptive Server 可能要将大量事务日志记录重新应用于数据页。

请参见 《系统管理指南，卷 2》中的第 11 章 “制定备份和恢复计划”，了解有关设置恢复间隔的信息。 sp_sysmon 报告检查点的数量和持续时间。请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

管家清洗任务对恢复时间的影响

Adaptive Server 的管家清洗任务在服务器空闲周期自动开始清理脏缓冲区。如果该任务可以刷新所有已配置高速缓存中的所有活动缓冲池，它将唤醒检查点进程。这可能使得检查点进程更快，数据库恢复时间更短。

系统管理员可使用 housekeeper free write percent 配置参数来调优或禁用管家清洗任务。此参数指定管家任务可增加数据库写操作的最大百分比。

有关调优管家和恢复间隔的详细信息，请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

审计和性能

繁重的审计可影响性能，如下所示：

- 审计记录首先被写入内存的队列中，然后写入 sybsecurity 数据库。如果该数据库与其它繁忙数据库共享一个磁盘，则会降低性能。
- 如果内存中审计队列被填满，则生成审计记录的用户进程转入休眠状态。请参见第 112 页的图 5-5。

确定审计队列的大小

审计队列的大小可由系统安全员设置。缺省配置如下：

- 单个审计记录最小需要 32 字节、最大需要 424 字节的空间。

  这意味着单个数据页存储 4 到 80 个记录。

- 审计队列的缺省大小为 100 个记录，需要大约 42K。

  该队列的最小大小为 1 个记录；最大为 65,335 个记录。

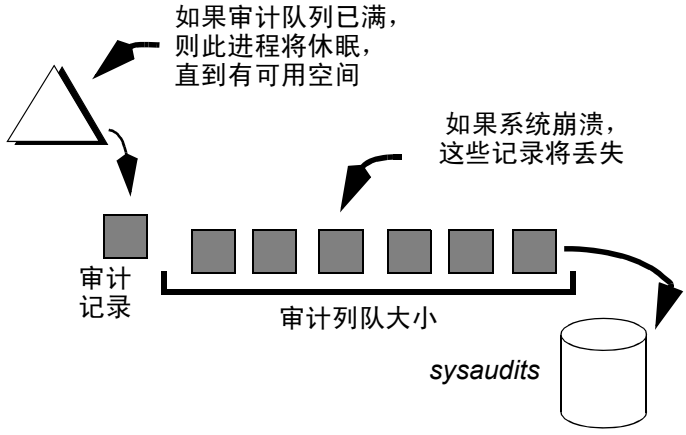
确定审计队列大小时需要进行权衡，如图 5-5 中所示。

审计队列较大时，没有用户进程休眠的风险，但会有在系统出现故障时损失内存中所有审计记录的风险。可能丢失的最大记录数为可存储在审计队列中的最大记录数。

如果安全是您关心的主要问题，那么应保持较小的队列。如果能够承担丢失更多审计记录的风险，且需要高性能，那么可使用较大队列。

增加内存中审计队列的大小会占用分配给数据高速缓存的总内存的一部分。

图 5-5：审计中的权衡和性能



审计性能准则

- 繁重的审计会降低整个系统性能。仅审计需要跟踪的事件。
- 如可能，将 sysaudits 数据库放在其自己的设备上。如果不能，请将其放到不用于最关键应用程序的设备上。

文本页和图像页

Text 和 image 页会占用大量内存，并且通常被认为会浪费空间。只要父数据行指向 text 和 image 页，这种浪费就始终存在。当对列执行 null update 后，这些页就会出现。

查找表的当前状态：

```
sp_help table_name
```

使用 sp_chcattribure 可解除分配 text 和 image 页，以释放它们占用的空间：

```
sp_chgattribute table_name, “deallocate_first_txtpg”,1
```

这将启用解除分配功能。要禁用解除分配功能，请输入：

```
sp_chgattribute table_name, “deallocate_first_txtpg”,0
```

第 6 章

调优异步预取

本章将解释异步预取如何在查询需要之前将数据和索引页预先读入高速缓存，从而改善多种类型查询的 I/O 性能。

异步预取如何改善性能

异步预取通过预测数据库活动 （对其来说，访问模式是可预测的）的某些明确定义的类所需要的页来改善性能。在查询需要这些页之前发出对它们的 I/O 请求，这样大多数的页在查询处理需要访问它们时已经处于高速缓存中。异步预取可以改进以下性能：

- 顺序扫描，例如表扫描、聚簇索引扫描和被覆盖的非聚簇索引扫描
- 通过非聚簇索引进行的访问
- 某些 dbcc 检查和 update statistics
- 恢复

对于访问大量页的查询，例如决策支持应用程序，在计算机的 I/O 子系统未达到饱和状态的情况下，异步预取可以提高查询性能。

如果 I/O 子系统已饱和或 Adaptive Server 是 CPU 密集型，则异步预取不能发挥作用 （或只能发挥少量作用）。异步预取可用于某些 OLTP 应用程序，但由于 OLTP 查询通常执行的 I/O 操作较少，其利用率很低。

当 Adaptive Server 中的查询需要执行表扫描时，它：

- 检查页中的各行和每行的值。
- 检查高速缓存寻找要从表中读取的下一页。如果页存在于高速缓存中，则继续处理任务。如果不在，任务将发出一个 I/O 请求并进入休眠状态直到此 I/O 操作完成。
- I/O 操作完成后，该任务将从休眠队列移到运行队列。在引擎上调度任务后， Adaptive Server 将检查新获取的页上的行。

这种对磁盘读取操作的执行和暂停循环将一直持续下去，直到表扫描完成。类似地，使用非聚簇索引的查询同样需要处理数据页，为索引所引用的下一页发出 I/O 命令，如果该页不在高速缓存中则进入休眠状态直到此 I/O 完成。

这种先执行然后等待 I/O 的模式将降低对大量页发出物理 I/O 的查询的性能。除了完成物理 I/O 所需的等待时间外，任务还重复开启和关闭引擎，从而增加了处理开销。

通过预取页改善查询性能

异步预取在查询需要某些页之前发出对它们的 I/O 请求，从而多数页在查询处理需要访问它们时已经处于高速缓存中。如果所需页已存在于高速缓存中，查询不会放弃引擎来等待物理读取。查询也可能会因其它原因放弃，但这种情况较少。

根据所执行的查询的类型，异步预取将建立它预计很快就会需要的页的**预见性设置**。 Adaptive Server 为使用异步预取的每一处理类型定义了不同的预见性设置。

某些情况下，预见性设置是非常精确的；但有时某些假定和推测可能会使预取的页永远不会被读取。当读入高速缓存的页中不需要的页所占的比例很小时，异步预取所获得的性能远远超过了读取无用页的影响。如果未用页的数量变大， Adaptive Server 检测到这种情况后，会减小预见性设置的大小或临时禁用预取。

多用户环境下的预取控制机制

当许多同时发生的查询将大量页预取到缓冲池后，为某个查询预取的缓冲区在使用前可能会有从缓冲池中刷新的危险。

Adaptive Server 将跟踪由异步预取送入每个缓冲池中的缓冲区及被使用的数目。它对已预取但未使用的缓冲区维护着一个以每个缓冲池为基础的计数。缺省情况下， Adaptive Server 会将异步预取限制设置为每个缓冲池的 10%。另外，对那些预取但未使用的缓冲区的数量限制可以以每个缓冲池为基础进行配置。

缓冲池限制和使用情况统计信息充当异步预取管理器，来保持较高的高速缓存命中率并减少不需要的 I/O。总体而言，结果可确保多数查询都能有较高的高速缓存命中率，仅有较少的查询会由于磁盘 I/O 休眠而导致延迟。

以下部分描述如何为使用异步预取的活动和查询类型构造预见性设置。在某些异步预取优化中，使用分配页来建立预见性设置。

有关分配页如何记录对象存储相关信息的信息，请参见 《性能和调优系列：物理数据库调优》中的第 2 章 “数据存储”。

恢复过程中的预见性设置

恢复过程中， Adaptive Server 读取每个包含事务记录的日志页，然后读取该事务引用的所有数据和索引页，以检验时间戳并回退或前滚事务。然后，对下一已完成的事务执行同样的操作，直到数据库中的所有事务都得到处理。两个独立的异步预取活动可加快恢复：对日志页本身的异步预取和对所引用的数据及索引页的异步预取。

预取日志页

事务日志按顺序存储在磁盘上，填充每一分配单元的扩充。每次恢复进程从新分配单元读取日志页时，它都将预取该分配单元中由日志使用的所有页。

在没有独立日志段的数据库中，日志和数据扩充可能混合在同一个分配单元中。异步预取仍预取该分配单元的所有日志页，但预见性设置可能会变小。

预取数据和索引页

对每一事务， Adaptive Server 都将扫描日志，然后从每个引用的数据和索引页建立预见性设置。在处理一个事务的日志记录时，异步预取会对日志中后续事务所引用的数据和索引页发出请求，并在处理当前事务之前读取这些页。

注释 恢复只使用缺省数据高速缓存中的缓冲池。有关详细信息，请参见第 125 页的 “为恢复设置限制”。

顺序扫描过程中的预见性设置

顺序扫描包括表扫描、聚簇索引扫描和被覆盖的非聚簇索引扫描。

在表扫描和聚簇索引扫描中，异步预取将使用用于对象的页的有关分配页信息来构造预见性设置。每次从新的分配单元读取一页，预见性设置是从对象使用的分配单元中的所有页中构造的。

顺序扫描在分配单元之间跳转的次数用来测量页链的分段。此值用来调整预见性设置的大小，以便在分段较低时预取大量页，而在分段较高时预取少量页。请参见第 120 页的 “页链分段”。

非聚簇索引访问中的预见性设置

当使用非聚簇索引访问行时，异步预取查找非聚簇索引叶页上所有限定的索引值的页号。它从所需的所有页的唯一列表中建立预见性设置。

仅当有两个或两个以上的限定行时才使用异步预取。

如果一个非聚簇索引访问需要多个叶级页，也会对这些叶页发出异步预取请求。

dbcc 检查过程中的预见性设置

在下列 dbcc 检查过程中使用了异步预取。

- dbcc checkalloc，它检查数据库中所有表和索引的分配以及相应的对象级命令 dbcc tablealloc 和 dbcc indexalloc
- dbcc checkdb，它检查数据库中的所有表和索引链接， dbcc checktable 检查独立的表及其索引

分配检查

dbcc 命令 checkalloc、 tablealloc 和 indexalloc （检查页分配）将验证有关分配页的信息。对分配进行检查的 dbcc 操作的预见性设置与其它顺序扫描的预见性设置方式类似。当扫描进入对象的一个不同分配单元后，就从对象使用的分配单元的所有页中建立预见性设置。

checkdb 和 *checktable*

dbcc checkdb 和 dbcc checktable 命令检查表的页链，以与其它顺序扫描相同的方式建立预见性设置。

如果正在检查的表中存在非聚簇索引，则从根页开始沿指向数据页的所有指针以递归方式对这些索引进行扫描。当检查从叶页到数据页的指针时， dbcc 命令以与非聚簇索引扫描类似的方式使用异步预取。当某个叶级索引页被访问后，预见性设置将从叶级索引页所引用的所有页的页 ID 中建立。

预见性设置的最小和最大值

查询在给定时间点的预见性设置大小由以下几个因素决定：

- 查询类型，如顺序扫描或非聚簇索引扫描
- 查询引用的对象所使用的缓冲池大小和每个缓冲池的预取限制设置
- 在执行扫描操作的情况下，表或索引的分段
- 异步预取请求的当前成功率和 I/O 请求的过载条件及服务器的 I/O 限制

表 6-1 总结了不同类型的异步预取用法中的最小和最大值。

表 6-1：预见性设置大小

访问类型	操作	预见性设置大小
表扫描 聚簇索引扫描 覆盖叶级扫描	从一个新的分配单元读取一页	查询至少需要 8 页 最大值是以下数值中的较小值： • 属于某一对象的分配单元上的页数 • 缓冲池预取限制
非聚簇索引扫描	定位叶页上的限定行并准备访问数据页	最小值是 2 个限定行 最大值是以下数值中的较小值： • 叶索引页上限定行中的唯一页号的数量 • 缓冲池的预取限制
恢复	恢复一个事务	最大值是以下数值中的较小值： • 正在恢复的事务所接触到的所有数据和索引页 • 缺省数据高速缓存中缓冲池的预取限制
	扫描事务日志	最大值是属于日志的分配单元的所有页
dbcc tablealloc、 indexalloc 和 checkalloc	扫描页链	与表扫描相同
dbcc checktable 和 checkdb	扫描页链	与表扫描相同
	检查链接数据页的非聚簇索引	叶级页引用的所有数据页。

当预取被自动禁用时

异步预取会尝试将所需页获取到缓冲池中，但不释放缓冲池或 I/O 子系统，并且不读取不需要的页。如果 Adaptive Server 检测到预取页正被读入高速缓存中但并未使用，则它将临时限制或停止异步预取。

释放缓冲池

对于数据高速缓存中的每个缓冲池，异步预取可读入可配置的缓冲区百分比并保持到缓冲区的首次使用时。例如，如果一个 2K 的缓冲池有 4000 个缓冲区，并且缓冲池的限制为 10%，则异步预取最多可读入 400 个缓冲区并在缓冲池中维持未使用状态。如果缓冲池中未访问的预取缓冲区数量达到 400，Adaptive Server 将临时停止对该缓冲池的异步预取。

如果查询访问了缓冲池中的页，池中未使用的缓冲区数量减小，异步预取即可恢复操作。如果可用缓冲区数量少于预见性设置中的缓冲区数量，则只发出前一数量的异步预取。例如，如果在允许有 400 个未用缓冲区的缓冲池中已有 350 个未用缓冲区，而某个查询的预见性设置是 100 页，则只发出前 50 个异步预取。

这可避免多个异步预取请求在能被读取之前用刷新高速缓存中的页的请求来释放缓冲池。由于每个缓冲池限制而未能发出的异步 I/O 数量由 sp_sysmon 报告。

I/O 系统过载

Adaptive Server 和操作系统为服务器整体和每一引擎设置未完成 I/O 数量的限制。配置参数 max async i/os per server 和 max async i/os per engine 控制 Adaptive Server 的这些限制。

有关为硬件配置 I/O 的详细信息，请参见操作系统文档。

配置参数 disk i/o structures 控制 Adaptive Server 保留的磁盘控制块的数量。I/O 队列中的每个物理 I/O（每个缓冲区读或写）都需要一个控制块。

请参见 《系统管理指南，卷 1》中的第 5 章 “设置配置参数”。

如果 Adaptive Server 尝试发出将超过 max async i/os per server、 max async i/os per engine 或 disk i/o structures 的异步预取请求，它将发出足以达到该限制的请求并放弃剩余的请求。例如，如果只有 50 个磁盘 I/O 结构可用，而服务器试图预取 80 页，则只发出 50 个请求，其它 30 个被放弃。

sp_sysmon 命令报告异步预取超出限制的次数。请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

尝试调优系统以便没有延迟的 I/O。如果有因以下原因而延迟的 I/O：

- 磁盘 I/O 结构，请增加 number of disk i/o structures 配置参数
- 服务器或引擎限制，请增加最大 max async i/os per engine 和 max async i/os per server 配置参数。
- 操作系统，请调优操作系统以便它可以处理更多的并发 I/O。

不必要的读取

异步预取尽量避免不必要的物理读取。在恢复和非聚簇索引扫描中，预见性设置是精确的，只读取在事务日志或索引页中由页号引用的页。

表扫描预见性设置，聚簇索引扫描以及 dbcc 检查更为随机，可能导致不必要的读取。顺序扫描中，以下因素可导致不必要 I/O 操作：

- 所有页锁定表的页链分段
- 多用户使用的大量高速缓存

页链分段

Adaptive Server 页分配机制在已分配给对象的扩充上分配新页并在已由对象使用的分配单元上分配新扩充，从而尽量使属于同一对象的页在物理存储上相互靠近。

然而，当分配和释放页时， DOL 锁定表上的页链可产生扭结。图 6-1 显示了一个位于两个分配单元的扩充之间的扭结页链示例。

图 6-1：跨越分配单元的页链中的扭结

在图 6-1 中，当扫描首次需要访问分配单元 0 中的页时，它将检查分配页并对其扫描的对象所使用的所有页发出异步 I/O，最高达到设置的缓冲池限制。当高速缓存中的页变为可用时，查询将根据页链次序对其进行处理。当扫描到第 10 页时，页链中的下一页 （第 273 页）属于分配单元 256。

当需要第 273 页时，将检查分配页 256 并对属于该对象的分配单元中的所有页发出异步预取请求。

当页链指回到分配单元 0 中的某页时，有两种可能：

- 来自分配单元 0 的预取页仍在高速缓存中，查询继续处理所需的物理 I/O。
- 来自分配单元 0 的预取页已被来自分配单元 256 的读取和使用缓冲池的其它查询所发出的 I/O 从高速缓存中刷新。查询必须重新发出预取请求。有两种方法可检测这种状态：
  - 分配页间的 Adaptive Server 的跳转计数现在为 2。Adaptive Server 使用跳转计数和预取页之间的比例来减小预见性设置的大小，因此可使发出的 I/O 较少。
  - 缓冲池中已预取但未使用的页的计数可能会很高，因此根据缓冲池的限制，可能会将暂时停止或减少异步预取。

异步预取的调优目标

为缓冲池选择最佳池大小和预取百分比是通过异步预取来获得性能改善的关键。当多个应用程序同时运行时，经良好调优的预取系统可平衡缓冲池大小和预取限制，从而实现以下目标：

- 改善系统吞吐量
- 提高使用异步预取的应用程序的性能
- 不降低未使用异步预取的应用程序的性能

对缓冲池大小和预取限制的配置更改是动态的，允许进行更改以适应不同工作量的需求。例如，可在恢复过程中或 dbcc 检查过程中配置异步预取以获得良好的性能，并且以后重新配置时无需重新启动 Adaptive Server。

请参见第 125 页的 “为恢复设置限制”和第 125 页的 “为 dbcc 设置限制”。

配置命令

异步预取限制被配置为已预取但未使用的页在缓冲池中能够储存的百分比。有两个配置级别：

- 服务器范围的缺省值，使用配置参数 global async prefetch limit 设置。安装 Adaptive Server 时， global async prefetch limit 的缺省值为 10 (%)。
- 每缓冲池覆盖，使用 sp_poolconfig 设置。若要查看为每个缓冲池设置的限制，请使用 sp_cacheconfig。

请参见 《系统管理指南，卷 1》中的第 5 章 “设置配置参数”。

异步预取限制的更改无需重启而立即生效。可以在配置文件中配置全局限制和每个缓冲区限制。

Adaptive Server 的其它性能特征

此部分涉及异步预取和其它 Adaptive Server 性能特征的交互。

大 I/O

大 I/O 和异步预取的组合可以较低的 I/O 开销为执行表扫描的查询和 dbcc 操作提供快速查询处理。

当大 I/O 预取一个分配单元的所有页后，整个分配单元的 I/O 数量的最小值是：

- 31 个 16K I/O
- 在与分配页共享一个扩充的页上为 7 个 2K I/O

注释 对大 I/O 的引用位于一台逻辑页大小为 2K 的服务器上。如果服务器的逻辑页大小为 8K，则用于 I/O 的基本单位为 8K。如果服务器的逻辑页大小为 16K，则用于 I/O 的基本单位为 16K。

16K 缓冲池的大小和限制

以 10% 的缺省异步预取限制执行 31 个 16K 预取需要至少有 310 个 16K 缓冲区的缓冲池。如果缓冲池较小或限制值较低，则某些预取请求将被拒绝。若要允许缓冲池中有更多的异步预取活动，请配置较大的缓冲池或为缓冲池配置较大的预取限制。

如果多个重叠查询使用同一缓冲池执行表扫描，则缓冲池中允许的未使用预取页数量需要更高一些。查询可能在略有交错的时间段及读取所访问页的不同阶段发出预取请求。例如，一个查询可能恰好预取了 31 页，且在缓冲池中有 31 个未使用页，而前一个查询只剩余两三个未使用页。要启动对这些查询的调优工作，假定以预取请求页数的一半乘以缓冲池中活动查询的数量。

2K 缓冲池的限制

在顺序扫描中使用大 I/O 的查询可能仍需要执行 2K I/O：

- 当扫描进入一个新的分配单元后，它将在与分配页共享空间的 7 个页上执行 2K I/O。
- 如果发出预取请求时分配单元中的页已经在 2K 缓冲池中，则必须将共享该扩充的页读入 2K 缓冲池。

如果 2K 缓冲池的异步预取限制设置为 0，则前 7 个读取以正常异步 I/O 执行，如果页不在高速缓存中，查询将在每次读取期间休眠。在不降低预取性能的情况下，将 2K 缓冲池的限制设置得足够高。

读取及放弃 (MRU) 扫描

当一个扫描使用 MRU 替换策略时，将以一种特殊方式处理由异步预取读入高速缓存的缓冲区。首先，页被链接到链的 MRU 末端而不是链接到清洗标记。查询访问页时，缓冲区在清洗标记处重新链接到缓冲池。此策略有助于避免清洗标记处所链接的预取缓冲区在使用之前被大量使用的高速缓存所刷新。它对性能的影响很小，除非预取了大量未使用页。在这种情况下，预取页很可能会刷新高速缓存中的其它页。

并行扫描和大 I/O

并行查询时对缓冲池的需求可能会变高。对相同缓冲池进行串行查询操作时，可以合理地假设，发出查询的时间略有不同且各查询处于执行的不同阶段：某些查询正在访问已在高速缓存中的页，其它查询正在等待 I/O。

并行执行对缓冲池有不同的需求，它取决于扫描类型和并行度。某些并行查询很可能会同时发出大量的预取请求。

基于散列的表扫描

所有页锁定表上的基于散列的表扫描具有全部访问同一页链的多个工作进程。每个工作进程都将检查表中每一页的页 ID，但只检查页 ID 与工作进程的散列值相匹配的页上的行。

需要从新分配单元获得页的第一个工作进程对该单元的所有页发出预取请求。当其它工作进程的扫描也需要该分配单元的页时，这些扫描会发现其所需的页已在 I/O 或高速缓存中。当第一个扫描完成要进入下一单元时，进程重复进行。

在执行基于散列的表扫描的进程系列中，只要有一个工作进程没有被延迟（如等待一个锁定），则基于散列的表扫描就不会对缓冲池发出比对串行进程更高的需求。由于多进程读取页的速度比串行进程要快很多，所以它们可更快地将页由未使用状态变为已使用状态。

基于分区的扫描

由于多工作进程可能对不同的分配单元执行异步预取，基于分区的扫描很可能会产生对缓冲池的额外需求。在多个设备的已分区表上，很少能达到每个服务器和每个引擎的 I/O 限制，而每个缓冲池的限制则更有可能限制预取。

并行查询被分析和编译后，它将启动自己的工作进程。如果一个有 4 个分区的表正被 4 个工作进程扫描，则每个工作进程都会试图预取其第一个分配单元中的所有页。对这一单个查询的性能而言，最理想的结果是 16K 缓冲池的大小和限制足够大，可允许 124 (31*4) 个异步预取请求，从而所有请求都可成功。每个工作进程快速扫描高速缓存中的页，然后移动至新的分配单元并对大量页发出更多的预取请求。

异步预取限制的特殊设置

出于特殊目的可能需要暂时更改异步预取配置，这些特殊目的包括：

- 恢复
- 使用异步预取的 dbcc 操作

为恢复设置限制

在恢复过程中，Adaptive Server 只使用缺省数据高速缓存中的 2K 缓冲池。如果使用 shutdown with nowait 关闭服务器，或服务器由于电源或机械故障而关闭，则需要恢复的日志记录数量可能会很大。

若要加快恢复速度，请编辑配置文件来实现以下一个或两个目的：

- 通过减小缺省数据高速缓存中其它缓冲池的大小来增加 2K 缓冲池的大小
- 增加 2K 缓冲池的预取限制

这两种配置更改是动态的，所以可使用 sp_poolconfig 在恢复完成后恢复初始值，而无需重启 Adaptive Server。只要 master 数据库恢复完成，恢复进程就允许用户登录服务器。数据库每次恢复一个，并且对于特定数据库，只要完成恢复，用户就可开始使用。如果某些数据库正在恢复中，并且在缺省数据高速缓存中的 2K 缓冲池中有大量的用户活动，就会产生争用。

为 *dbcc* 设置限制

如果在执行数据库一致性检查时服务器上的其它活动很少，对 dbcc 使用的缓冲池配置较高的异步预取限制可加快一致性检查。

如果相应的数据库在高速缓存中没有 16K 缓冲池， dbcc checkalloc 可使用特殊的内部 16K 缓冲区。如果某个数据库有 2K 缓冲池而没有 16K 缓冲池时，在执行 dbcc checkalloc 命令时应将该缓冲池的本地预取限制设为 0。使用 2K 缓冲池而不是 16K 内部缓冲区可能会降低性能。

高预取性能的维护活动

如果修改表中的数据，所有页锁定表和叶级索引的页链将产生扭结。通常，新建表很少扭结。由于更新、删除和插入而导致页面拆分、新页分配和页面解除分配的表很可能会产生跨分配单元的页链扭结。如果已修改表中超过 10% 至 20% 的初始行，即应确定扭结的页链是否降低了异步预取效率。如果怀疑页链扭结降低了异步预取性能，可能需要通过重新创建索引或重新载入表来减少扭结。

消除堆表中的扭结

对于所有页锁定堆，页分配一般是有顺序的，除非由于删除了页中的所有行而释放了页。当为该对象分配了其它空间后，这些页可以被重新使用。可创建一个聚簇索引（如果希望将表存储为堆，则删除它）或将数据批量复制出来，然后截断表，并再次复制数据。每个活动都将压缩表使用的空间并消除页链扭结。

消除聚簇索引表中的扭结

对于聚簇索引，页面拆分和页面解除分配可导致页链扭结。要消除所有跨分配页扭结，并不一定要重建聚簇索引。对预期将要增长的聚簇索引，可使用 fillfactor 来减少由于数据修改而导致的扭结数量。

消除非聚簇索引中的扭结

如果查询混合使用了覆盖索引扫描，则当页级页链变为碎片后，删除和重建非聚簇索引可改善异步预取性能。

性能监控和异步预取

statistics io 的输出报告由异步预取执行的物理读取和由常规异步 I/O 执行的读取的数量。另外， statistics io 还报告高速缓存中由未持有高速缓存螺旋锁的异步预取找到的页的搜索次数。

请参见 《性能和调优系列：利用统计分析提高性能》中的第 1 章“使用 set statistics 命令”。

sp_sysmon 报告在 “Data Cache Management”部分和 “Disk I/O Management”部分都包含有关异步预取的信息。

如果使用 sp_sysmon 来评估异步预取性能，可能会看到其它性能方面的改善，如：

- 在异步预取发挥作用的池中高速缓存命中率将更高。
- 环境切换由于高速缓存未命中而相应减少，同时主动放弃增加。
- 可能的锁争用减少。在为查询所需的下一页执行 I/O 期间，任务将保持对页的锁定。如果由于异步预取增加了高速缓存命中率而使此时间变短，则锁定的保持时间也将变短。

请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

索引

符号

数字

英文

B

C

D

F

G

H

J

K

L

M

N

P

Q

R

S

T

W

X

Y

Z