软件的性能设计

The Performance Design of Software

崔晓东* 曹彤宇
CUI Xiao-dong CAO Tong-yu

摘 要 本文概要介绍了大型软件系统性能设计的误区，指出性能设计应该是设计过程的一个必备环节。根据实际的性能设计经验，作者提出了性能设计的基本要求和主要方法。

关键词 软件 性能设计 架构设计

Abstract This article summarily introduced the misunderstanding for the performance design of a large software system, and pointed out that the performance design should be a necessary step in the design process. The authors discussed the fundamental requirements and major methods about performance design, according to their practice experiences.

Keywords Software Performance Design Architecture Design

1 序言

对于大型应用系统而言，系统的性能显得极为重要。这里的性能主要指稳定性和运行速度。功能再强，而性能很差，频繁宕机，系统便失去了可用性。因此，性能设计是软件中必不可少的组成部分。本文结合笔者在大型 B/S架构应用系统的设计经验，谈一谈软件性能设计的基本方法。

2 性能设计的误区

关于软件的性能设计，通常有两个误区：

第一个误区：性能问题可以依靠更好的计算机硬件和系统软件来解决。

在设计系统前，系统设计师得到的信息可能是：不用担心软硬件问题，我们用的都是最好的。而实际可能的结果是：内存不够就加大，PC服务器不行就换小型机，Windows操作系统不行就改为 Linux或者 UNIX，但出现的性能问题却不一定有显著的改观。

事实上这种做法是很盲目的，要知道无论什么编程语言都可以写出让系统崩溃的程序来。

第二个误区：系统的性能可以通过系统优化来实现。

通常的做法是先不管性能，尽快实现功能，等到系统测试或试运行的时候再调优。其实，这个做法是不可取的。因为系统优化是一种事后行为，依靠的是数据库、应用服务器的调优能力，虽然能在一定程度上解决问题，但是一旦遇到它们也无能为力的情况，我们就需要花费巨大的代价来解决。

这两个误区产生的后果往往都是事后补救的时候才发现无可奈何，因为没有事先建立起性能问题的预防机制。

有句话说得好：设计使然！有什么样的设计就有什么样的结果，没有设计就别指望会有满意的结果。好的性能也是设计出来的。因此真正可取的做法是在软件设计的过程中完成性能设计，而不仅仅是在测试或试运行的过程中调试，测试过程或试运行过程应当是检验性能设计效果的过程。

3 性能设计的基本要求

孙子兵法云：知己知彼，百战不殆。

首先，必须了解性能目标。

性能目标源于用户需求，但这些需求往往是隐藏的或是潜在的需求，如果没有被识别出来，可能就会被设计师所忽略。

对于大型的 B/S架构的应用系统，设计师应注意弄清楚以下一些基本问题：

3.1 数据形态、数据规模和增长速度

数据是以文件形式居多还是以数据库形式居多？这决定了数据的访问效率，以及是否需要专门的文件服务器或数据库服务器。

数据是集中存储还是分散存储？集中存储有利于集中管理，但服务器的压力较大，这时需要重点考虑服务器的运行效率。而分散存储虽然可以缓解服务器压力，但是会增加数据同步的难度，影响数据交换的质量，这时需要提高设计质量以避免数据损失。

总的数据量在怎样的数量级？记录数是十万、百万、千万还是更多？这个数量级决定了应该选择什么样的数据库系统，以及需要在多大程度上考虑性能设计。记录数较少，一般的数据库服务器都可以轻松应付，但是对于千万级的记录量，就要综合运用

* 上海南康科技有限公司 200030

多种设计方法了。

数据的增长速度怎样？每天或每月会以怎样的一个数量级增长？这个数量级决定了应重点设计性能的哪个方面。对于生产型系统（以采集数据为主），数据的增长速度较快，应重视设计其存储效率；对于决策支持型系统（以统计分析为主），数据常以静态方式存在，应重点设计其检索效率。数据的增长速度还决定了系统在多久后必须考虑扩容，以及应该准备什么样的备份策略。

3.2 用户的数量

设计师应当清楚系统有多少现实的和潜在的用户。用户数量在一定程度上决定了系统现实的压力情况和未来可能的压力情况。如果用户数量上千甚至上万，就要重视在应对访问压力方面的性能设计了。

3.3 系统的并发访问量

系统运行速度变慢甚至宕机往往是发生在并发访问量较高的时候。因此，有必要弄清楚并发访问量发生在每天的哪个时段，最高能够达到多少，突发情况会达到多少。这几个数值是性能设计的依据，也是性能测试的依据。

对于一个生产型系统而言，有 24小时持续压力的，但是多数都是 8—12小时的。如果某几个时段压力过大，可以考虑让用户分时段使用系统，将压力分摊到不同时段去。

3.4 网络状况

网络状况常常被忽视。通常，在开发环境下，网络状况都是理想的，网速很快很稳定。但是在实际环境下并非如此，因为网络上需要部署各类防火墙、入侵检测、防病毒等软硬件，网速和稳定性上会受到一定程度的影响。在理想条件下做出的设计可能不会满足实际的需要。

如果网络状况不好，意味着用户不见得能够正常完成系统的某些操作。这时，设计师应加强系统的容错性设计，避免数据的异常丢失。

其次，要清楚地了解系统运行的软、硬件环境。

这些环境就如同战场上的地形，如果对地形不熟悉就不容易发挥地形的优势，弄不好还会陷入不利境地。如果系统需要在不同的环境下运行，那么还必须注意环境之间的差异。

具体地说，设计师应注意以下关于运行环境的问题：

3.5 操作系统

在 Java平台上，比较常见的开发方式是：在 Windows操作系统上开发，然后移植到 Linux或者 Unix操作系统上运行。前者可以保证开发效率，后者则可提高安全性和运行效率，但是应事先了解操作系统之间的差异，否则会遭遇难以逾越的障碍。

比如，很多 Linux和 Unix操作系统都不安装窗口系统，甚至完全是西文的。如果你在 Windows环境下所做的设计恰好用到了窗口系统的功能（如图形、字体），移植到这样的 Linux或 Unix环境下就会出现异常。

此外，它们与 Windows操作系统默认的字符集也不同，这会影响到文件的读写。

3.6 数据库

如果要设计一个适用于不同数据库类型（如 Oracle和 SQL Server）的系统，可移植性是设计师必须面对的问题，特别要注意这些数据库之间的差异，在 SQL语法和所使用的函数方面可能会引起问题。如果确实必须用到它们的特性，就有必要提供一个集中处理这种差异的机制。

另外一个需要重视的是数据库的字符集。使用中文字符集还是西文字符集，如果标准不统一可能会导致乱码出现，这就是一个正确性的问题了。

3.7 应用服务器

应用服务器（如 WebLogic，Tomcat）虽然原理上基本相同，但性能上却各有千秋。

在选定应用服务器类型的情况下，应注意发挥其优势，充分利用其性能调优的功能，同时避免其短处，不要让系统的缺陷影响系统性能。

再好的应用服务器，也是人一行一行代码写出来的，因此，它难免存在一些缺陷。很多应用服务器的补丁一打再打，多数是解决各类性能问题的。设计师应了解这些缺陷，从而避免掉入“陷阱”。

再次，要掌握必要的设计方法和辅助工具。

“工欲善其事，必先利其器”，借助常用的设计方案和辅助工具，可以使得设计师的工作更有效。对于一些常见的性能问题，都有一些通用的设计方案，就如同下棋中的“定式”，了解这些方案有助于提高工作效率。

计算机技术发展至今，对于常见的具体问题，已经积累了大量成熟的算法、组件和工具，设计师要能够活学活用这些知识，把它们组合起来，发挥更大的威力。

同时，也要求设计师具有一定的分析和计算能力。

其实，运算速度是可以计算出来的。众所周知，快速排序算法在大多数情况下比冒泡排序算法更快，这不光是实验的结果，还是理论计算的结论。

这要求设计师具备一定的数学知识，具备一定的推理能力。在这一点上不像数学本身那样严格要求准确，只要理论值接近实际值即可，这样可以估计出系统在真实环境下的性能情况。

4 性能设计的主要方法

有了性能目标和运行环境，剩下的就是我们如何发挥主观能动性，借助“地利”，给出创造性的设计方案了。

4.1 要消除影响性能的结构性问题

前面分析了可能影响系统性能的各种因素，有些只影响到系

统的局部，而有些则会影响全局。这些影响全局的因素会造成结构性问题，这些因素必须被首先消除。

结构性问题是致命的，就如同一幢楼房，如果整体结构上有问题，不但会有大厦将倾的风险，甚至根本无法建立起来这个系统。关于这一点，在运行环境和系统压力方面尤其要注意。

对于影响系统性能的因素，可以按照重要程度从高到低排序，优先解决那些重要程度较高的问题。

4.2 要分析并找出可能的性能瓶颈

这些瓶颈是性能设计的关键部位，通常是在局部发挥作用。应着重找出那些被频繁访问或运行复杂数据处理的部分，改进这些部分设计可以达到事半功倍的效果。

在 B/S架构的应用系统中，以下情况可能产生性能瓶颈：

● 数据库中关联的表过多（如超过 5个）或者一次要求返回的记录数太多（如超过 1000条）可能会在数据库运算方面产生瓶颈，因为表的关联会导致指数级增长的笛卡儿积运算，遍历太多的记录也比较费时。

● 应用服务器中的共享资源（如数据库连接池、线程池、缓存）在高并发条件下可能会产生访问冲突的瓶颈。在 Java语言中，有比较著名的“线程死锁”，应用服务器会因此宕机。在高并发条件下，应借助一些辅助方法，才能有效解决这个问题。

● 一次请求返回的数据量太大（比如超过 2兆字节）可能会产生网络传输的瓶颈，速度会明显变慢。如果数据量确实较大，则需要考虑使用压缩技术。

4.3 将特定的性能问题分解到不同的系统组件来解决

在典型的 B/S架构下，数据需要流经客户端、网络、应用服务器、数据库服务器等诸多环节，任何一个环节压力过大都可能造成阻塞。

一般地，当客户端发出一个请求，用户可以接受的等待时间也就是 3到 5秒钟。任何阻塞都可能会延长等待时间。因此，对于那些频繁使用的功能，系统应在数据流经的每个环节上进行优化设计，以提高总体响应速度。

笔者就曾经遇到这样一个例子。某个大数据量的统计功能，设计师把计算功能完全交给数据库，写了 400多行的 SQL语句，耗时数分钟才给出计算结果。后改为在应用服务器中执行两条简单的 SQL语句并组装数据，数据库服务器仅负责基本数据的提取，应用服务器负责数据的转换，整个过程耗时只有几十毫秒，速度提高了好几个数量级。

“平衡才健康”，在性能设计中尤其如此。

4.4 建立模型

做性能设计也需要做试验，越是规模大的系统越是需要这个步骤。试验的对象就是为此建立的模型。

模型是基于理论创建的，它瞄准的是系统的设计目标，依托的是系统的运行环境，并综合运用了各种数据结构、通用的或定制的算法和处理模式，因此是为特定系统量身订做的，我们甚至可以计算出模型定量的性能指标，在理论上具有一定的可靠性。

对于软件系统，已经有很多成熟的模型可供参考。但是正如世界上找不到两片相同的叶子一样，每个软件系统都有其个性，那些成熟模型只能作为局部的参考，需要在其基础上进一步改进。

实践是检验真理的唯一标准，对于建立的模型还需要不断地做试验。试验的目的就是检验模型能够在多大程度上满足性能设计要求，从而进行不断地改进。这样的模型才是稳定可靠的，是能够经得起考验的。

你可能会问：现在软件开发追求的是效率——以最快的速度完成系统建设，如果试验这样做下去，系统什么时候才能上线啊？

事实证明：磨刀不误砍柴工。真实的运行环境不是试验场所，那时再出问题，延误的将不仅仅是工期！

4.5 应避免掉入系统软件固有缺陷的“陷阱”

任何软件都会有长处，设计师应设法让这些长处得到充分发挥。任何软件也都难免有缺陷，设计师要做的是预防这些缺陷发挥作用，如果控制不好会有适得其反的结果。

笔者在这方面吃过苦头。我们在 2006年设计上海市房屋土地资源管理局某大型 B/S架构的业务系统时，上线运行时发生断断续续的宕机情况。由于“相信”系统软件的性能“应该不会有问题”，我们一直从自身设计上找原因。

这个系统的用户量超过一万，数据量是千万记录级的，还跟其他几个系统有接口，因此开始的时候我们怀疑是系统无法承受压力所致，所以不断优化算法，进行持续的压力测试，但耗时良久却状况依旧。事实上，经不断观察我们发现：系统在每次宕机前压力并不大，应该不是压力造成的。

最终我们把怀疑的目光放在系统软件上，原来是应用服务器软件的一个自身缺陷造成的，打上最新的补丁之后再也没有宕过机。我们花了四个月完成了系统的开发，却花了六个月才查出这个问题，这笔学费可真是够高的！

有鉴于此，设计师必须了解各种系统软件的优势和缺陷，扬长避短，才能达到良好的性能要求。

5 结束语

软件的性能设计是系统设计师的责任，性能设计是建设大型应用系统不可或缺的一步。有了性能设计，系统设计师就会对系统未来的性能胸有成竹，可以避免返工，减少浪费，提高软件质量。因此，系统设计师应重视性能设计，做好性能设计！

（收稿日期：2007－09－28）