**Google集群架构**

为了能够支持可扩展的并行化，google的网络搜索应用让不同的查询由不同的处理器处理，同时通过划分全局索引，使得单个查询可以利用多个处理器处理。针对所要处理的工作负载类型，google的集群架构由15000个普通pc机和容错软件组成。这种架构达到了很高的性能，同时由于采用了普通pc机，也节省了采用昂贵的高端服务器的大部分花费。

很少有网络服务的单个请求像搜索引擎占用那样多的计算资源。平均来看，在google上的每次查询需要读取数百m的数据耗费数10亿的cpu指令循环。为了能够支持峰值在每秒数千次请求流，需要与世界上最大的超级计算机规模相当的硬件设施。通过容错性软件将15000个普通pc机联合起来，提供了一种比使用高端服务器更廉价的解决方案。

本文我们将介绍google的集群架构，讨论那些影响到设计方案的最重要的因素：能效和性价比。在我们的实际操作中，能效实际上是一个关键的度量标准，因为数据中心的电力是有限的，因此电力耗费和制冷成为运作中的关键。

我们的应用本身可以很容易进行并行化：不同的查询可以运行在不同的处理器上，同时全局索引也划分的使得单个查询可以使用多个处理器。因此，处理器的性价比比峰值性能变得更重要。同时，google的应用是面向吞吐率的，可以更有效的利用处理器提供的并行化，比如并行多线程(SMT)，或者多核处理器(CMP)。

**Google架构概览**

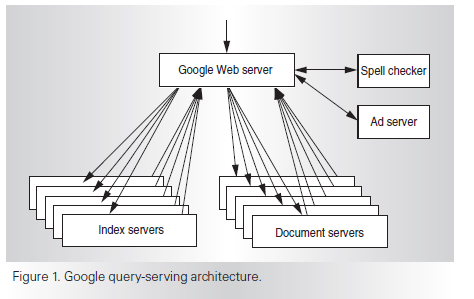
Google的软件架构来源于两个基本的观点。首先我们需要在软件层面提供可靠性，而不是通过硬件，这样我们就可以使用普通的pc构建廉价的高端集群。其次，我们不断的裁剪设计是为了达到最好的总体请求吞吐率，不是为了提高服务器的峰值响应时间， 因为我们可以通过并行化独立的请求来控制响应时间。

我们相信使用不可靠的廉价pc来构建可靠的计算设施可以达到最好的性价比。通过在不同的机器上备份服务，以及自动化的故障检测和错误处理，为我们的环境提供软件级的可靠性。这种软件级的可靠性在我们的系统设计中几乎随处可见。检查一下一次查询处理的控制流程，有助于理解这种高级的查询服务系统，同时也有助于对于可靠性考虑的理解。

**Google的一次查询**

当用户在google中输入一次查询，用户浏览器首先通过DNS进行域名解析，将[www.google.com](http://www.google.com/)转换为ip地址。为了对查询可以进行更有效的处理，我们的服务由分布在世界各地的多个集群组成。每个集群大概有数千个机器，这种地理上的分布可以有效的应付灾难性的数据中心失败比如地震，大规模的停电。基于DNS的负载平衡系统，会计算用户的与每一个物理集群地理上的距离来选择一个合适的物理集群。负载平衡系统，需要最小化请求往返时间，同时要考虑各个集群的可用容量。

用户浏览器然后给这些集群中的一个发送一个http请求，之后，对于该集群来说，所有的处理都变成了本地化的。在每个集群中有一个基于硬件的负载平衡器监控当前可用的google web servers(GWS)集合，并在这个集合上将本地的请求处理进行负载平衡。收到一个请求之后，GWS协调这个查询的执行，并将结果格式化为html语言。图1表示了这个过程。



查询执行由两个主要阶段组成，第一个阶段，索引服务器查阅倒排索引(将每个查询词映射到匹配的文档列表)。索引服务器然后决定相关的文档集合，通过对每个查询词匹配的文档列表求交集，为每个文档计算出一个相关性的分值，这个分值决定了在输出结果中的排序。

 搜索的过程非常具有挑战性，因为需要处理海量数据：原始网页文档通常具有数十T的未压缩数据，从原始数据中导出的倒排索引本身也有好几T的数据。幸运的是，通过将索引划分到不同的片段，可以将搜索高度并行化，每个片段具有从全布索引中随机选择的一个文档子集。一组机器负责处理对于一个索引片段的请求，在整个集群中每个片段都会有这样的一组机器与之对应。每个请求通过中间负载平衡器选择组内机器中的一个，换句话说每个查询将会访问分配到每个片段的一台机器(或者是一组机器的子集)。如果一个片段的备份坏了，负载平衡器将会避免在查询时使用它，我们的集群管理系统的其他组件将会尝试修复它，实在不行就用另一台机器来取代它。停工期间，系统的容量需要减去那台坏掉的机器所代表的容量。然而，服务仍然是未中断的的，索引仍然是可用的。

第一阶段的查询执行最终输出一个排过序的文档标识符列表。第二阶段则通过获取这个文档列表，然后计算出所有文档的标题和url以及面向查询内容的文档摘要。文档服务器处理这项任务，从硬盘中获取文档，抽取标题以及查询关键词在文档中的出现片段。像索引查找阶段，这里的策略也是对文档进行划分，主要通过：随机分布文档到不同的小片段；针对每个片段的处理具有多个服务器作为备份；通过一个负载平衡器分发请求。文档服务器必须能够访问一个在线的低延时的整个网络的网页的副本。实际上由于对于这个副本的访问需要性能及可用性，所以google实际上在集群中存储了整个web的多个副本。

除了索引和文档服务阶段，GWS在收到查询时还会初始化几个其他的辅助任务，比如将查询发送给拼写检查系统，广告系统生成相关广告。当所有阶段完成后，GWS生成html输出页面，然后返回给用户浏览器。

**使用备份进行容量扩充和容错**

我们对系统进行了一些结构化以保证对于索引和其他响应查询相关的数据结构是只读的：更新是相对不频繁的，这样我们就能通过将查询转移到一个服务备份来安全的进行更新。 这条原则，使得我们避免了很多在通用数据库中出现的一致性问题。

我们也尽力挖掘出大量应用中的固有的并行性：比如我们将在一个大索引中的匹配文档的查询转化为针对多个片段中的匹配文档的多个查询加上开销相对便宜的归并步骤。类似的，我们将查询请求流划分为多个流，每个由一个集群来处理。增加为每个处理索引片段的机器组增加机器来增加系统容量，伴随着索引的增长增长片段的个数。通过将搜索在多个机器上并行化，我们降低了响应一个查询的必需的平均延时，将整个计算任务划分在多个cpu和硬盘上。因为独立的片段相互之间不需要通信，所以极速比几乎是线性的。换句话说，单个索引服务器的cpu速度不会影响整个搜索的整体性能，因为我们可以增加片段数来适应慢的cpu，因此我们的硬件选择主要关注那些可以为我们的应用提供出色的请求吞吐率的机器，而不是可以提供最高的单线程性能的那些。

简单来说，google的集群主要遵循下面三个主要设计原则：

软件可靠性。我们没有选择硬件性容错，比如采用冗余电源，RAID，高质量组件，而是专注于软件可靠性。

使用备份得到更好的吞吐率和可用性。因为机器本身是不可靠的，我们备份我们的内部服务在很多机器上。通过备份我们得到了容量，与此同时也得到了容错，而这种容错几乎是免费的。

性价比重于峰值性能。我们购买当前最具性价比的cpu，而不是那些具有最高绝对性能的cpu。

使用普通pc降低计算花费。这样我们可以为每一个查询提供更多的计算资源，在ranking算法中使用更昂贵的技术，可以搜索文档的更大的索引。

**使用商业化部件**

Google的机柜是专门定制的，由两面组成，总共放置了40到80个基于80x86的服务器(每侧包含20个20u或者40个10u服务器) 。我们对于性价比的偏爱，使得我们选择自己组装的桌面pc通过它们的组件，除了选择大的硬盘驱动器。目前的服务中使用了好几个cpu产品，从533m intel-celeron 到双核1.4G Intel奔三服务器。每个服务器包含一个或者多个IDE硬盘，每个80g空间。与文档服务器相比，索引服务器通常具有更少的磁盘空间，因为它的负载类型是对cpu更敏感。在机柜一侧的服务器通过一个100m 以太网交换机相连，该交换机通过一个或者两个GB级的链路与一个核心的GB交换机相连，该核心交换机连接所有的机柜。

我们的根本选择标准是单次查询花费，可以表示为性能/资金花费总和(包括折旧)+管理花费(主机，系统管理，维修)。实际来看，一个服务器的寿命通常不会超过2，3年，因为与新机器相比，无法在性能上保持一致。三年前的机子性能上要远远落后于当前的机子，对于包含这两类机器的集群，很难达到合适的负载分布和配置。有了这个相对的短期分摊周期，可以看到设备的花费在总的开销中占到相当的一部分。

因为google服务器是专门定制的，我们可以使用基于pc的服务器机柜价格做一个展示。比如在2002年，一个88 个双核cpu 2G intle xeon，2G内存，80G硬盘的机柜在RackSaver.com上的价格大概是27800$,转换成三年周期每月的花费将是7700$。剩下的主要花费是人力和hosting。

 设备花费的相对重要性使得采用传统的服务器解决方案并不适合解决我们的问题，因为虽然它们可以提高性能，但是降低了性价比。比如4处理器的主板是很昂贵的，由于我们的应用已经进了很好的并行化，这样的主板不需要额外的花费就能获得很好的性能.类似的，尽管SCSI硬盘更快也更可靠，但是它们通常比同样容量的IDE硬盘贵2-3倍。

使用基于基于廉价的PC的集群比使用高端多处理器服务器在成本上的优势是非常明显的，至少对于像我们这样的高并行化应用来说。上面例子中的27800$的机柜包含176个2G Xeon CPU，176G内存，7T硬盘空间。与此相比，一个典型的x86服务器具有8个2GCPU，64G内存，8T硬盘空间，花费大概758000$。换句话说，这个服务器贵了3倍，但是cpu只是原来的1/22，内存是1/3,硬盘空间稍微多了点。价格的差距，主要是源于更高的互联网络带宽和可靠性。但是google的高度冗余架构并不依赖于这两个中的任何一个。

管理数千台中型PC机和一些高端多处理器服务器将会带来完全不同的系统管理和维修费用。然而，对于一个相对同构的的应用来说，大部分的服务器只用来运行少数应用中的一个，这些开销是可管理的。加速安装和更新工具都是可用的，维护1000台和100台服务器所需的时间和成本相差并不大，因为所有的机器都具有相同的配置。类似的，通过使用可扩展的应用监控系统，监控的花费伴随这集群的大小增长也不会有太大的增长。另外，我们可以通过批处理修复将修复的开销保持在一个较低的水平，同时保证我们可以很容易的替换掉那些具有高损坏率的组件，比如硬盘和电源。

**电力问题**

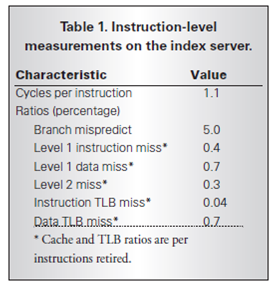
如果没有特殊的高密度包装，电力消耗和制冷设备将会成为一个挑战。一个中型的1.4G奔三服务器在有负载情况下，通常要耗费90w直流电：55w给cpu，10w给硬盘，25w给DRAM加电和主板，对于一个ATX电源，通常具有75%的效率，这样转化成交流电就是120w，每个机柜就需要10kw。一个机柜需要25 ft2 的空间，这样算下来，用电的密度就是400w/ft2。如果采用高端处理器，一个机柜的用电密度可以超过700w/ft2。

 不幸的是，通常的商业数据中心电力密度在70-150w/ft2之间，远远低于pc集群的需求。这样，如果低端pc集群使用相对直接的包装方式，就需要特殊的制冷和额外的空间来降低用电密度使得与标准数据中心兼容。因此只要机柜还存放在标准数据中心中，如果想要往机柜里增加服务器就会在实际部署中收到限制。这种情况就使得我们考虑降低单服务器的电力使用是否是可能的。

 对于大规模集群来说，低功耗服务器是非常具有吸引力的。但是我们必须要牢记以下几点：降低电力是迫切的，但是对于我们的应用来说，不能带来性能上的惩罚，我们关心的是每单元性能的瓦特，不是单纯的瓦特，第二，低功耗的服务器必须不能太过昂贵，因为服务器的折旧花费通常要超过电力的花费。前面提到的10kw机柜，每月大概消耗10mwh的电力(包括制冷的开销)，假设每kwh电15美分，这样每月需要花费1500$，与折旧的7700$相比并不算大。因此低功耗服务器不能够多于我们通过采用常规pc节省下的那部分花费。

**硬件级别的应用特点**

检查我们应用的各种架构特点可以帮助我们搞清楚哪种硬件平台可以为我们的索引查询系统提供最高的性价比。我们着重于索引服务器的特点，该模块的性价比对于整体的性价比起着主导性的作用。索引服务器的任务主要包括：对倒排索引中的压缩数据进行解压，找到与一个查询相匹配的文档集合。表1展示了一些索引服务器程序的基本的指令级别的度量，程序运行在一个1G双核奔三系统上。



   考虑到奔三每个circle基本上可以处理三条指令，可以看到该应用程序的CPI(cycles per instruction)稍微偏高。我们可以预见到这种行为，考虑到我们的应用程序使用了很多动态数据结构而控制流是依赖于数据的，这样就会产生大量的很难预测的分支。事实上，如果相同的工作负载在奔四处理器上运行时，CPI几乎增长了2倍，分支预测几乎相同，尽管奔四具有更强大的指令并行和分支预测功能。可见，在这种工作负载类型下，并没有太多的指令级并行性可供挖掘。测量结果显示，对于我们的应用来说，出现在处理器中的大量的乱序不确定性执行成为降低程序性能的关键点。

 对于像索引服务器这样的应用来说，更合适的挖掘并行性的方式应该是提高平凡的计算并行性。系统在处理每个查询时共享只读数据，建立只需要很少通信的工作单元。我们在集群级别上通过部署大量的廉价节点取代少量的昂贵节点来发挥这个优势， 挖掘在微架构级别上的线程级并行性看起来也是可行的。并行多线程(SMT)和多处理器架构(CMP)都是面向线程级的并行性，都可以大大提高我们的服务器的性能。一些针对Intel Xeon处理器的早期实验表明通过使用两个上下文的SMT比单个上下文具有30%的性能提升。

 我们相信对于CMP系统提升的潜力应该是更大的。在CMP的设计中，采用多个简单的，按序执行的，短流水线核取代复杂的高性能核。如果我们的应用具有很少的指令级并行性(ILP)，那么由于按序执行所带来的性能惩罚也是很小的。同时短流水线将可以减少甚至排除分支预测失败所造成的影响。线程级的并行性伴随着核的增长可以呈现出接近线性的加速比，同时一个合理大小的共享L2 cache将会加速处理器间的通信。

**内存系统**

表1 也描述了主存系统的性能参数，我们可以观察到对于指令cache和指令tlb具有良好的性能，由于使用了较小的内层循环代码。索引数据块不具有时间局部性，因为索引数据大小变化剧烈同时对于索引的数据块访问模式是不可预测的。然而对于一个索引数据块的访问可以从空间局部性上获益，这种局部性能够通过硬件预取或者大的缓存line开拓出来。这样如果使用相对合适cache大小就可以得到好的全局cache命中率。

 内存带宽看起来并不会成为一个瓶颈。我们估计奔腾系列处理器系统的内存带宽使用率可以很好的控制在20%以下。主要是由于对于索引数据的每个缓存行，需要放到处理器cache里，这需要大量的计算资源，此外在数据获取中还存在天然的依赖关系。在很多情况下，索引服务器的内存系统行为正如TPC-D(Transaction Processing Performance Counicil’s benchmark D)所报告的那样。对于这种工作负载类型，采用一个相对合适的L2cache大小，短的L2 cache和内存延时，长的(比如128字节)cache line可能是最有效的。

**大规模多处理**

正如前面提到的，我们的设备是一个由大量廉价pc组成的庞大集群，而不是少数大规模的共享内存机组成的。大规模共享内存机主要用于在计算通信比很低的时候，通信模式或者数据划分是动态或者难预测的，或者总的花费使得硬件花费显得很少的时候(由于管理日常费用和软件许可证价格)。在这些情况下，使得它们的高价格变得合理。

 在google，并不存在这样的需求，因为我们通过划分索引数据和计算来最小化通信和服务器间的负载平衡。我们自己开发需要的软件，通过可扩展的自动化和监控来降低系统管理的日常费用，这些使得硬件花费成为整个系统开销中显著的一块。另外，大规模的共享内存机器不能很好的处理硬件或者软件的失败。这样大部分的错误可能导致整个系统的crash。通过部署大量的多处理器机，我们可以将错误的影响控制在一个小的范围内。总的来看，这样的一个集群通过明显的低成本解决了我们的服务对于性能和可用性的需求。

 初看起来，好像很少有应用具有像google这样的特点，因为很少有服务需要数千台的服务器和数pb的存储。然而可能有很多的应用需要使用基于pc的集群架构。一个应用如果关注性价比，能够运行在不具有私有状态的多个服务器上(这样服务器就可以被复制)，它都有可能从类似的架构中获益。比如一个高容量的web服务器或者一个 计算密集型的应用服务器(必须是无状态的)。这些应用具有大量的请求级并行性(请求可以划分在在独立的服务器上处理)。事实上，大的web站点已经采用这样的架构。

 在google规模上，大规模服务器的并行化的一些限制确实变得明显起来，比如商业数据中心在制冷容量上的限制，当前的cpu对于面向吞吐率的应用所做的优化还远远不够。虽然如此，通过使用廉价pc，明显地提高了我们可以为单个查询所能支付的计算量。因此有助于帮助我们提高成千上万的用户的网络搜索体验。

**致谢**

在这些年里，很多人为google的硬件架构做出了重要的贡献。在这里，特别感谢Gerald Aigner ，Ross Biro,Bogdan Cocosel 和Larry Page所做的工作